2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

UNITED STATES NAVY
NAVAL POSTGRADUATE SCHOOL

THESIS

Reusable Ada Software for Command and Control Workstation Map Manipulation

by

Bennett K. Larson

June 1992

Co-Advisor:	Patrick D. Barnes
Co-Advisor:	William G. Kemple

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable) CC	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
REUSABLE ADA SOFTWARE FOR COMMAND AND CONTROL WORKSTATION MAP MANIPULATION (U)

12. PERSONAL AUTHOR(S)
Larson, Bennett K.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 09/91 TO 06/92	1992, June, 18	205

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Reusable Software, Ada, Command and Control, Maps

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Current DOD Command and Control (C^2) priorities call for modular and interoperable C^2 systems that can be assembled easily from standard components to provide unique C^2 capabilities. Since one of the most costly and critical components of a C^2 system is its software, it makes sense to create reusable software components that can be used in this modular building process.

This thesis describes a reusable set of Ada software packages and a portable user interface that implements common two dimensional map and symbol manipulation functions in a C^2 workstation environment. This software provides students and researchers a graphical software environment within which different map-based C^2 system concepts such as situation monitoring and decision support can be developed and evaluated.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Patrick D. Barnes, CAPT, USAF	22b. TELEPHONE (Include Area Code) (408) 646-2830; 22c. OFFICE SYMBOL CS/Ba

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

REUSABLE ADA SOFTWARE FOR COMMAND AND CONTROL WORKSTATION MAP MANIPULATION

by
Bennett K. Larson
Captain, USAF
B.S., USAF Academy, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SYSTEMS TECHNOLOGY

from the

NAVAL POSTGRADUATE SCHOOL
June 1992

Author: Bennett K. Larson
Bennett K. Larson

Approved By: [illegible]
Patrick D. Barnes, Co-Advisor

[illegible]
William G. Kemple, Co-Advisor

[illegible]
Carl R. Jones, Chairman,
Command, Control, and Communications Academic Group

ABSTRACT

Current DOD Command and Control (C^2) priorities call for modular and interoperable C^2 systems that can be assembled easily from standard components to provide unique C^2 capabilities. Since one of the most costly and critical components of a C^2 system is its software, it makes sense to create reusable software components that can be used in this modular building process.

This thesis describes a reusable set of Ada software packages and a portable user interface that implements common two dimensional map and symbol manipulation functions in a C^2 workstation environment. This software provides students and researchers a graphical software environment within which different map-based C^2 system concepts such as situation monitoring and decision support can be developed and evaluated.

DISCLAIMER

Classic-Ada is a trademark of Software Productivity Solutions, Inc.

DECWindows is a trademark of Digital Equipment Corporation.

Frame Technology and FrameMaker are registered trademarks of Frame Technology.

ObjectMaker and Adagen are a registered trademarks of Mark V Systems Limited.

Open Look is a trademark of UNIX System Labs, Inc.

OSF/Motif is a trademark of the Open Software Foundation, Inc.

PostScript is a trademark of Adobe Systems, Inc.

Sun, SPARCStation, SunView and SunOS are registered trademarks of Sun Microsystems.

TAE is a trademark of the National Aeronautics and Space Administration.

UNIX is a trademark of Bell Laboratories.

VADS and Verdix are registered trademarks of Verdix Corporation.

X Window System and X11 are trademarks of the Massachusetts Institute of Technology.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

2D	Two Dimensional.
3D	Three Dimensional.
4GT	Fourth Generation Techniques.
ACM	Association for Computing Machinery.
AdaIC	Ada Information Clearinghouse.
ADRG	Equal Arc Second Raster Chart/Map Digitized Raster Graphics.
AFIT	Air Force Institute of Technology.
AJPO	Ada Joint Program Office.
ANSI	American National Standards Institute.
API	Application Programming Interface
ASSET	Asset Source for Software Engineering Technology.
ASR	Ada Software Repository.
ATW	Advanced Tactical Workstation.
BBS	Bulletin Board System.
BDA	Bomb Damage Assessment.
C^2	Command and Control.
C^2IS	Command and Control Information System.
C^3I	Command, Control, Communications, and Intelligence.
C^4I	Command, Control, Communications, Computers, and Intelligence.
CASE	Computer-Aided Software Engineering.
CD-ROM	Compact Disk-Read Only Memory.
CIA	Central Intelligence Agency.
CIM	Corporate Information Management.
CMS	Common Mapping System
CMU	Carnegie Mellon University, Pittsburgh, Pennsylvania.
COSMIC	Computer Software Management and Information Center.
DARPA	Defense Advanced Research Projects Agency.
DDN	Defense Data Network.

DMA	Defense Mapping Agency.
DOD	Department of Defense.
DSN	Defense Switching Network.
DTED	Digital Terrain Elevation Data.
DTIC	Defense Technical Information Center.
E-mail	Electronic Mail.
FACRP	Functional Analysis and Consolidation Review Panel.
FTP	File Transfer Protocol.
GAO	General Account Office.
GUI	Graphical user interface.
GIPSY	Graphic Information Presentation System.
I-CASE	Integrated Computer-Aided Software Engineering.
IEEE	Institute of Electrical and Electronics Engineers.
ISO	International Standards Organization.
J6	DOD Joint Commander Designation for the Director for Command, Control, Communications, and Computer Systems.
JOTH-T	Joint Over-The-Horizon Targeting.
JTLS	Joint Theater Level Simulation.
LCCDS	Low Cost Combat Direction System.
LCDR	Lieutenant Commander.
MAGIC	Mapping and Graphic Information Capability.
MIL-STD	Military Standard.
MIT	Massachusetts Institute of Technology.
MIVS	Multi-Source Integrated Viewing System.
NASA	National Aeronautics and Space Administration.
NORAD	North American Aerospace Defense.
NPS	Naval Postgraduate School.
NTIS	National Technical Information Service.
OOA	Object-Oriented Analysis.
OOD	Object-Oriented Design.
OS	Operating System.

OSF	Open Software Foundation.
PC	Personal Computer.
PEX	PHIGS Extension to X.
PHIGS	Programmer's Hierarchical Interactive Graphics System.
POSIX	Portable Operating System Interface.
RAPID	Reusable Ada Program for Information System Development.
RWDB2	Relational World Data Bank II.
SCCS	Source Code Control System.
SEI	Software Engineering Institute at CMU.
SLOC	Source Lines of Code.
SPARC	Scalable Processor Architecture.
SPC	Software Productivity Consortium.
StP	Software through Pictures.
STARS	Software Technology for Adaptable, Reliable Systems.
STSC	Software Technology Support Center.
TAE Plus	Transportable Applications Environment Plus.
TWX	Theater War Exercise.
UIMS	User Interface Management System.
USA	United States Army.
USAF	United States Air Force.
USGS	United States Geological Service.
USN	United States Navy.
USNR	United States Naval Reserve.
VADS	Verdix Ada Development System.
WDB II	World Data Bank II.
WPT	Window Programming Tools.
WVS	World Vector Shoreline.
WWMCCS	World Wide Military Command and Control System.
X	X Window System.
X11R4	X Version 11, Release 4.
X11R5	X Version 11, Release 5.

ACKNOWLEDGEMENTS

This work would not have been possible without the encouragement from my wife, Cindy, who always said I would have this work done on time. I also thank my thesis advisors for setting me on course and getting me back on it when I was losing my way. In addition, I thank those countless people on Usenet who provided me with great software examples and timely information. They made the research effort much more worthwhile.

Most of all I acknowledge the unfailing support and encouragement of the most important person in my life—Jesus Christ. I dedicate this work to Him.

I. INTRODUCTION

A recent Joint Chiefs of Staff review of DOD Command and Control (C^2) priorities calls for modular and interoperable C^2 systems that can be assembled easily from standard components to provide unique C^2 capabilities. The review document, *C^2 Functional Analysis and Consolidation Review Panel Report* (FACRP), recommends three key C^2 systems technology initiatives the military should pursue to meet commanders' operational needs. These initiatives focus on computer networking, command center integration, and research in "high pay-off" technologies such as artificial intelligence for information fusion and decision support [28, p. 40]. All of these initiatives are considered requirements for future C^2 systems, and all involve improving the current state of computer software.

These initiatives require specific actions to carry them out. The action items are outlined in a related letter written by the Assistant Secretary of Defense for Command, Control, Communications, and Intelligence (C^3I), Mr. Paul A. Strassmann. One of the software-related action items that is intended to help reduce C^2 system costs is to "concentrate software investments on reengineering, reuse, and retrofitting technologies." [55, p. 13] Action items dealing with system capabilities state that C^2 systems should be modular, interoperable, and portable to meet a wide range of needs [55, p. 3].

To accomplish these software-related action items, software tools are needed to develop reusable and portable software for future C^2 systems. This thesis describes the development of reusable Ada software for a core component of many C^2 systems—the component that displays cartographic (map) and track information. This reusable software is intended to be a tool for students and researchers for quickly creating a graphical software environment in which different map-based C^2 system concepts such as situation monitoring and decision support can be developed and evaluated.

A. WHY IMPROVED C^2 SOFTWARE IS NEEDED

Understanding the reasons for needing reusable and portable software for C^2 systems requires looking at relevant C^2 information issues from the perspectives of past events, current research results, the future vision for C^2 systems, and the financial costs. Although the specific issues addressed in this thesis are maps and the software that manipulates them on C^2

workstations, the following perspectives are not limited to map software issues. They do, however, emphasize the map-related information issues and needs of military commanders.

1. An Historical Perspective

The following three short accounts of past military use of force show how the absence of critical map-related information can seriously hurt U.S. forces.

A vivid account of how poor C^2 map-related information can cost many lives was the World War II battle for an obscure Pacific atoll called Tarawa. A cartographer for Admiral Nimitz, LCDR S. Ostermeier, USNR (retired), reported that information about the condition of the landing shores was woefully incomplete. During the initial assault, many landing craft became stranded on sand bars exposed by the low tides. The actual beach was still far away. The enemy easily killed scores of Marines due to this unfortunate circumstance. Better hydrographic knowledge (i.e. maps) and tide information could have helped the commanders make better decisions and probably would have saved many lives. [27]

A more recent, though less dramatic anecdote, told in several classes at the Naval Postgraduate School, about the U.S. military Grenada operation shows that access to cartographic information was still far from ideal in 1983. To find key targets in Grenada, many troops had to use tourist maps until better ones could be obtained. Even after the "better" maps arrived, they did not all use the same grid system. As a result, commanders could not order valuable fire-support for fear of hitting friendly troops.

Even more recently reported by the news media, Operation Desert Storm showed how important timely bomb damage assessment (BDA) is to a commander. Without accurate intelligence, some targets were unnecessarily revisited or mistakenly considered destroyed. In addition, more people than in the past were concerned about minimizing wartime casualties—both friendly and enemy—and demanded accountability for target selection and the application of appropriate means of force.

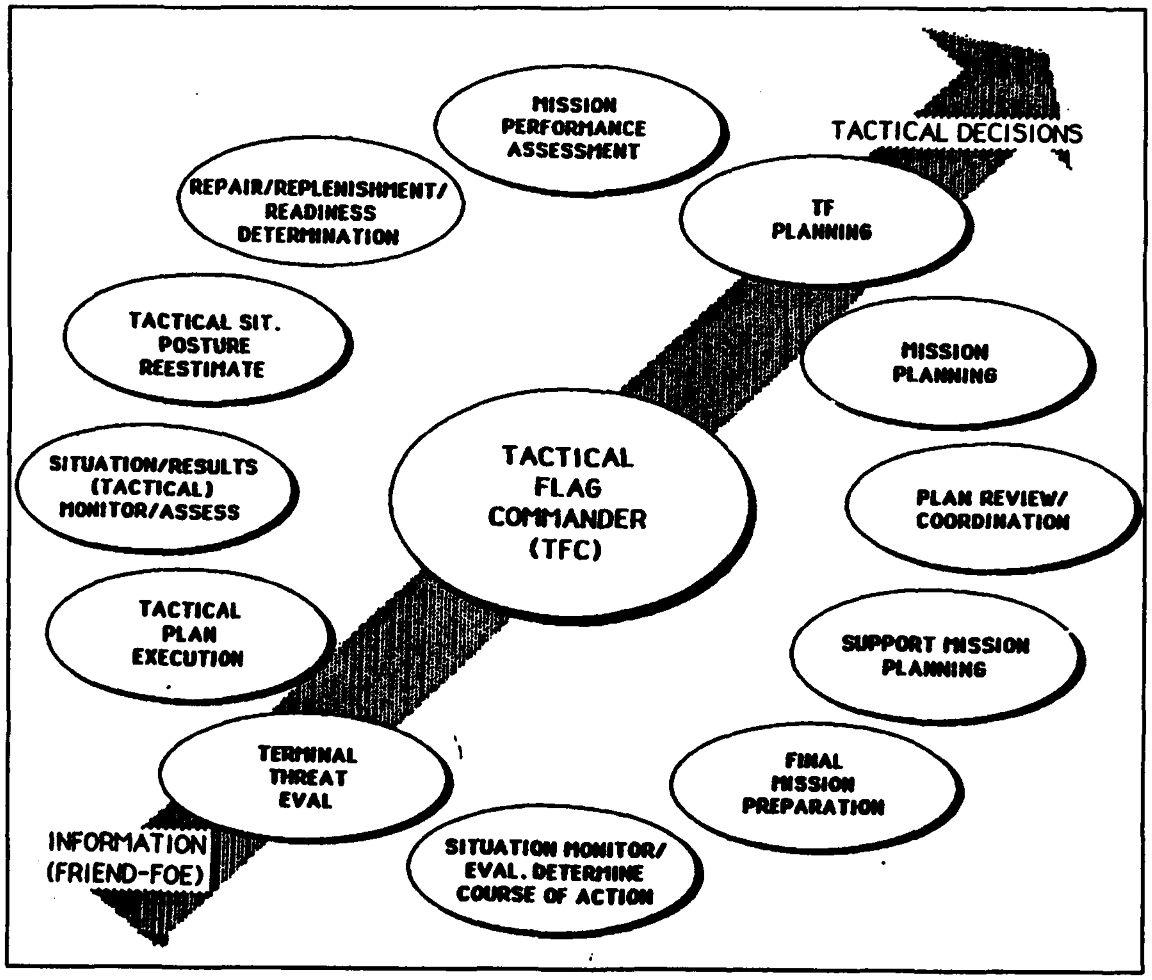
From an historical perspective, each of these accounts illustrates that there were and still are problems with gathering, disseminating, and displaying information that is often best communicated with maps. The problems related to getting timely map information to commanders are being addressed with new C^2 information systems based on current research.

2. A Current Research Perspective

Past problems in C^2 information distribution and processing demand a solution. Research into command decision-making, current problems commanders face, areas for applying decision aids and information systems, and technologies for implementing decision aids and information systems can help provide one.

a. Commander Decision-Making

The first step to finding a solution for this problem is to focus on the types of decisions a commander makes. As an example, Figure 1 shows the kinds of decisions a tactical flag commander has to make and the general direction taken to arrive at these decisions.



TACTICAL DECISIONS
MISSION PERFORMANCE ASSESSMENT
REPAIR/REPLENISHMENT/ READINESS DETERMINATION
TF PLANNING
TACTICAL SIT. POSTURE REESTIMATE
MISSION PLANNING
SITUATION/RESULTS (TACTICAL) MONITOR/ASSESS
TACTICAL FLAG COMMANDER (TFC)
PLAN REVIEW/ COORDINATION
TACTICAL PLAN EXECUTION
SUPPORT MISSION PLANNING
TERMINAL THREAT EVAL
FINAL MISSION PREPARATION
INFORMATION (FRIEND-FOE)
SITUATION MONITOR/ EVAL. DETERMINE COURSE OF ACTION

Figure 1. Tactical Flag Command Decision-Making [7, p. 37]

Stephen J. Andriole, a George Mason University C^2 systems engineering professor, developed this figure to show the range and complexity of tactical decision-making, but points out that commanders at the strategic and other levels of command in all military services also make similar kinds of decisions that need similar attention [7, p. 35].

b. *Current Decision-Making Problems*

Andriole lists several special problems commanders routinely face when trying to make decisions. He puts the problems that cause ineffective individual and group decision-making into two categories: Sub-optimal information management, and limited option generation and implementation. Sub-optimal information management includes difficulties in searching for data, information overloading, and poor information presentation. The limited option generation and implementation category includes such problems as limited real-time simulation capabilities and limited alternative generation [7, p. 35].

Figure 2 depicts these decision-making problems and suggests where C^2 information systems might fit to provide the most benefit [7, p. 38]. A database management system can organize the data and provide ways to quickly access critical data. A message system can handle incoming, outgoing, and internal communications—storing and filtering information so it can get to the right decision-makers. The central parts of an information system—the communications and operating system— provide the foundation for the database, message-handling and the applications software that provides the option generation and implementation capabilities.

c. *Application Areas that Address the Problems*

Several application areas have been the focus of effort to try and solve the commanders' most pressing problems. These application areas are shown in Figure 3. They cover a wide range of expertise pointing to the fact that the subject of C^2 is broad and complex. It is interesting to note that most of the same technologies are used to meet different needs. For instance, a map-based workstation display system could be used to help in decision-making, evaluation, option generation, and planning.

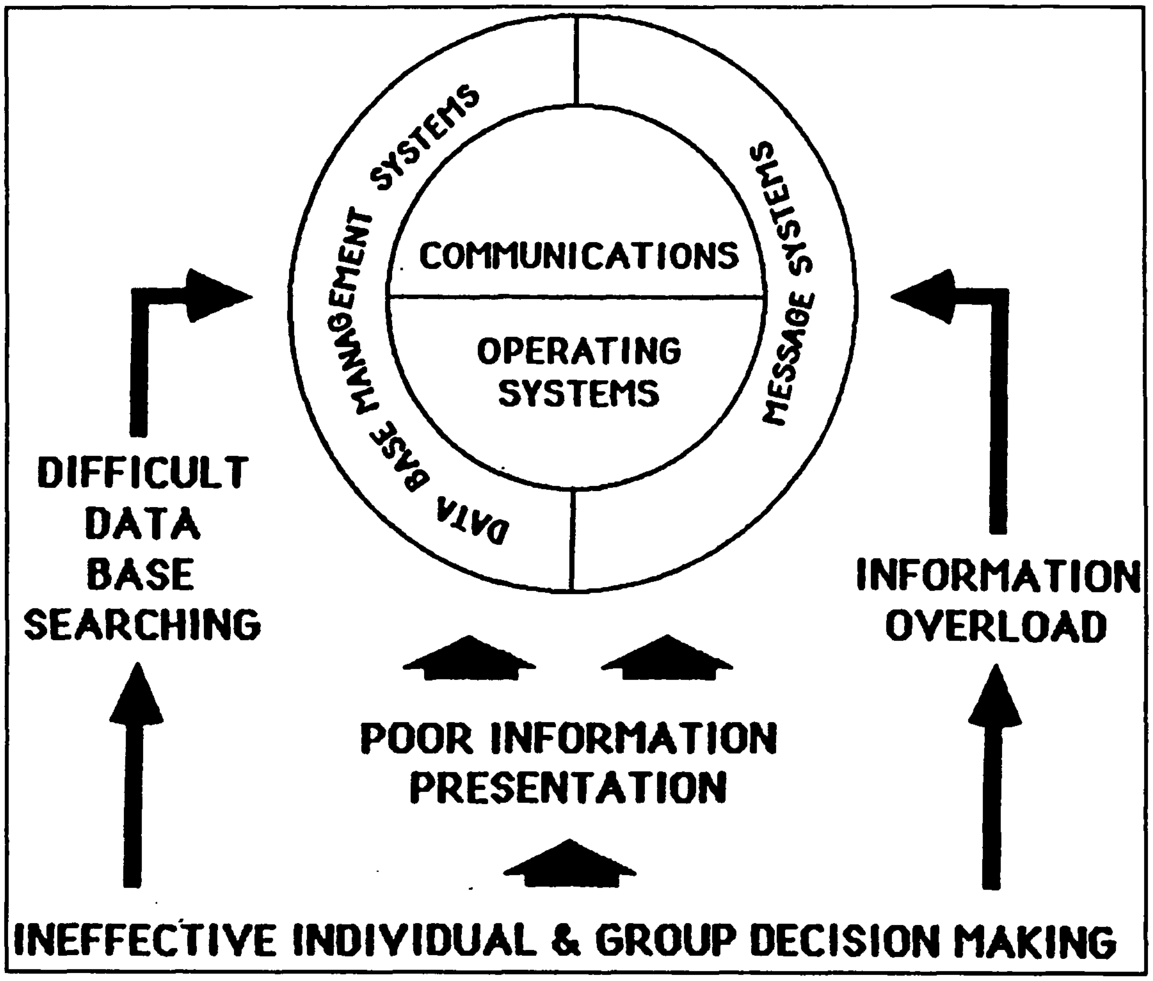

DATA BASE MANAGEMENT SYSTEMS
MESSAGE SYSTEMS
COMMUNICATIONS
OPERATING SYSTEMS
DIFFICULT DATA BASE SEARCHING
INFORMATION OVERLOAD
POOR INFORMATION PRESENTATION
INEFFECTIVE INDIVIDUAL & GROUP DECISION MAKING

Figure 2. C^2 Information Processing and Decision-Making Problems [7, p. 38]

d. Decision Aids and Information System Technologies

Three key technologies used to implement decision aids and information systems are shown in Figure 4. These technologies are analyses, database management, and user interface methods and tools. One general tool used for implementing many C^2 system user interfaces is the map/terrain display systems listed in Figure 4. In light of these tools, this thesis offers a sub-tool for developing map display systems.

To summarize this perspective, Andriole's research gives a framework for thinking of and developing systems to solve system information problems, and not a specific solution. The direction for specific solutions is being addressed by the current military leadership and is discussed in the next section.

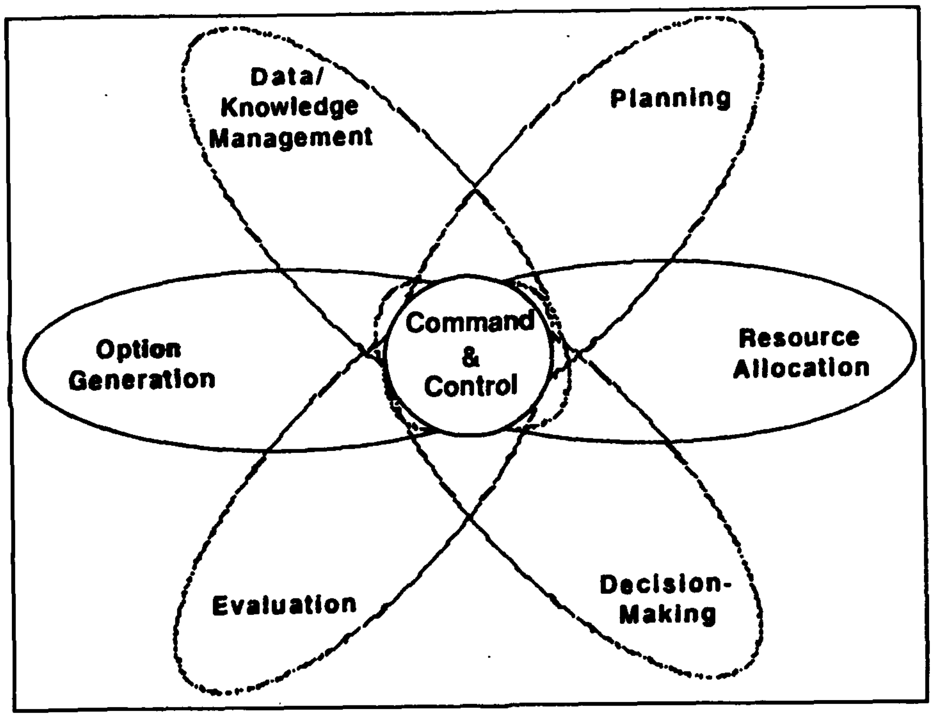


Figure 3. C^2 Requirements and Aiding Application Areas [7, p. 39]

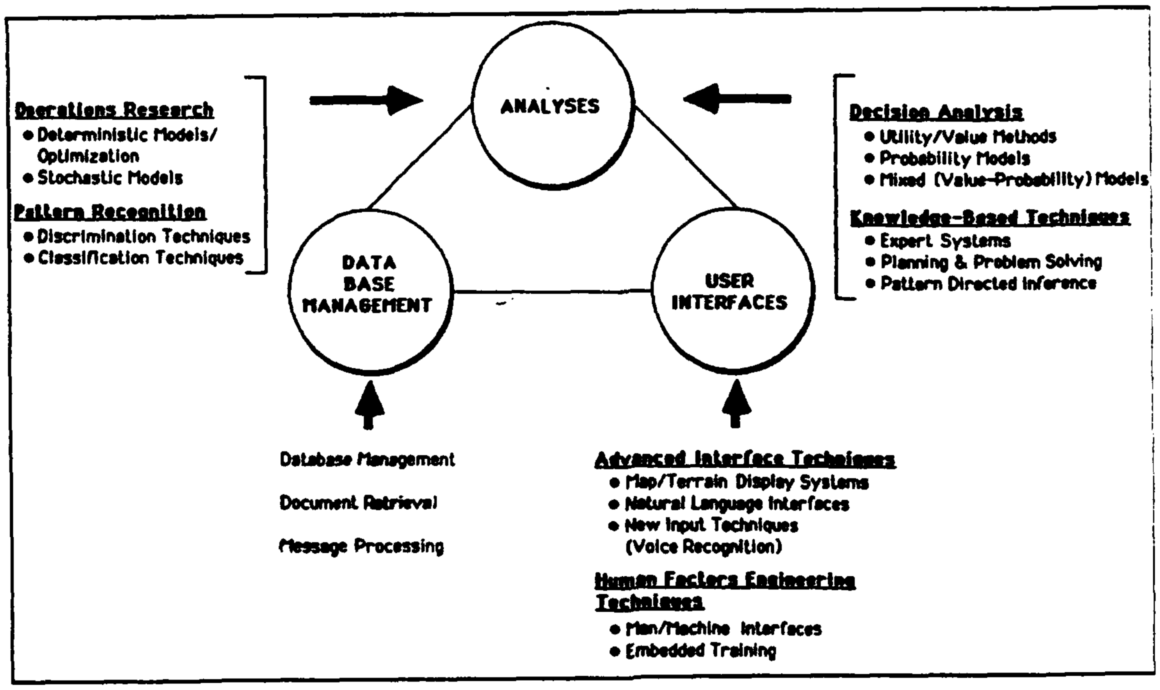


Figure 4. Major Information Processing and Decision-Aiding Technologies [7, p. 43]

3. A Forward-Looking Perspective

The current military leadership has developed a three phase overall plan to provide current and future commanders with effective systems. A "C^4I for the Warrior" briefing given to the Joint Chiefs of Staff by the office of the Director (J6) for Command, Control, Communications, and Computer Systems (C^4) outlined the phases as the "Quick Fix" solution, the mid-term solution, and the final objective. [29, p. 2]

The "Quick Fix" phase addresses immediate solutions over the next five years. The briefing stated that the biggest problem needing a solution is the lack of system interoperability (i.e. information sharing). This has occurred because each service has developed its C^2 system architectures independently, heading in its own direction [29, p. 7]. As a result these systems cannot work well together under current joint warfighting doctrine which calls for teamwork [20]. During the "Quick Fix" phase, interoperability problems are to be solved with hardware and software translators that can connect the incompatible systems. [29, p. 11]

The mid-term phase is concerned with solutions that have interoperability as a design feature. This phase extends from the end of the "Quick Fix" phase plus ten years. The primary design focus for all new systems will be interoperability. National and international standards will replace unique military standards to allow the construction of the final objective—an interoperable global information network for joint commanders.[29, p. 13]

As stated, the final objective, is to provide the joint commander the capability to connect to a global network of fused information. This phase is proposed for after the year 2000. By this time, it is envisioned that a commander should be able to access any needed information from anywhere and at anytime from one "joint warfighter terminal." [29, pp. 3,14]

From this briefing it is apparent that the senior-level perspective for C^2 systems includes mobile and highly configurable systems with workstation displays that provide flexibility for meeting future mission needs. Much of the flexibility and information capabilities for these future systems will be provided by the systems' software.

4. A Financial Perspective

As was shown in Figure 4, software is a major component of C^2 Systems. It is the software that manages the data, provides a meaningful information display, and helps the commander automatically analyze information and options. However, developing the software to provide these capabilities is expensive. Due to the lack of skilled workers relative to the software demand, costs are high. The total DOD software demand is currently estimated at $15 billion each year and is expected to keep rising steadily unless action is taken to change the situation. Figure 5 shows this trend [54]. Although this estimate is for all DOD software, the general trend is assumed to also hold true for C^2 software.

According to Strassmann [54], the cost of software can be controlled and eventually reduced through two coordinated efforts—software reuse and integrated computer-aided software engineering (I-CASE). The software reuse effort primarily focuses on reusable software repositories. Several DOD projects have been started to provide libraries of software that will facilitate the reuse of previously developed software. These projects will be mentioned in more detail in Chapter II. The second cost-saving effort targets I-CASE. In essence I-CASE is a managerial process that focuses on groupwork and cooperation primarily in software analysis and design, and includes what has been traditionally called CASE. When using I-CASE, the software cost savings are anticipated in the maintenance phase of the software lifecycle.

Figure 5 shows the estimated effect of implementing reuse and I-CASE. The top cost line shows the estimated future costs if no action is taken regarding the way software is currently being developed. The cost curve under the shaded region shows the effect of just reuse. It flattens out, but continues to rise. The line labeled as Achievable Software Costs shows the anticipated effect of reuse, I-CASE, and possibly other cost cutting measures combined. In this instance, the cost curve actually begins to decrease once maintenance cost savings start being realized. [54]

Trying to get a handle on software costs in the DOD has been an on-going effort. One of the most significant past efforts to help reduce DOD software costs was the development of the Ada programming language. In 1974 DOD software costs were $3 billion a year and cost overruns were blamed on the large number of programming languages that required expensive training and maintenance. Most of the problems seemed to center in embedded software for weapon systems. The decision was made to develop a standard language to program embedded systems. This language became known as Ada. [11] Soon after the language was accepted in 1983, it was realized there were significant benefits realized when using Ada in many non-

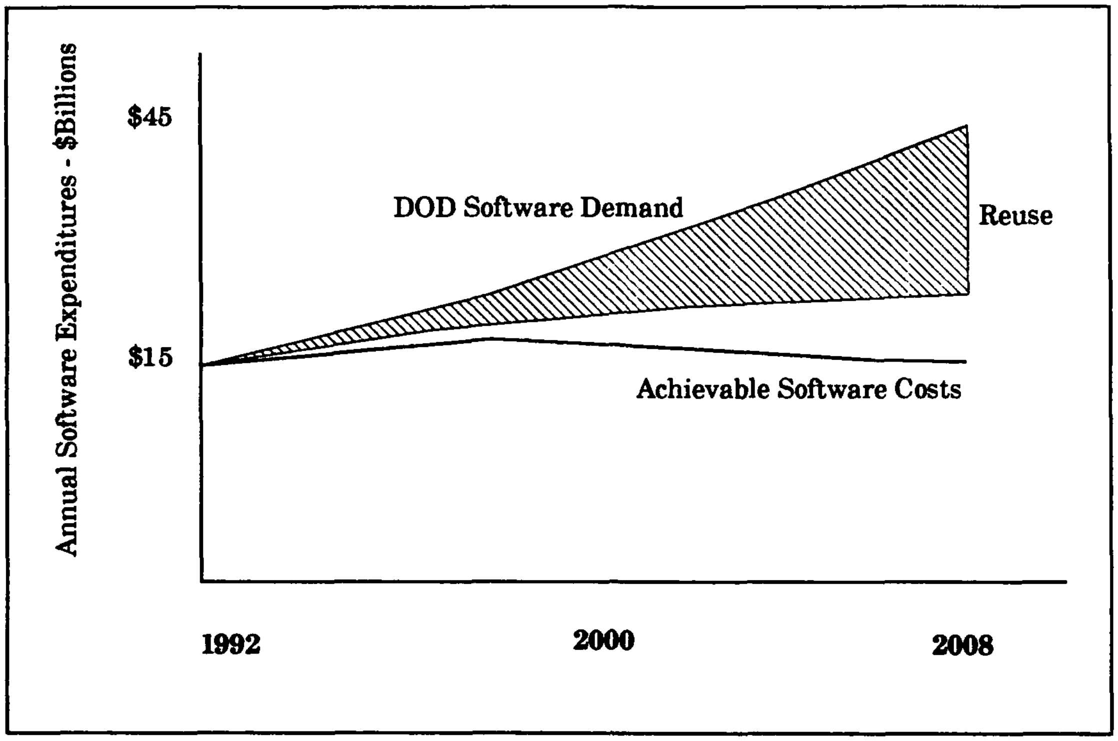


Figure 5. Estimate of DOD Achievable Software Costs [54]

embedded systems such as C^2 decision aids and information systems. One important benefit was that Ada provided many features that enforced good software engineering principles which other languages did not. Unfortunately, the cost-benefit of using Ada has been a difficult argument to quantitatively support.

In March 1989, when trying to analyze the cost-benefit of Ada, the General Accounting Office (GAO) concluded that the DOD was unable to produce information showing how Ada was helping to control software costs [67]. Since that time, the Ada Joint Program Office (AJPO) has kept a list of most of the major software development programs that are using Ada [4], but even this has not provided the necessary information for a good Ada cost-benefit estimate. However, on a smaller scale, a business case analysis was done comparing Ada to C++. According to this report, in general it was better to use Ada than C++ from an overall cost perspective. [3]

The financial perspective portrays DOD software development as requiring significant monetary resources. If the cost estimates are even somewhat accurate, it appears that software reuse and I-CASE are the logical paths to take for now.

5. Perspective Summary

The issues that appear in the historical, research, leadership, and financial perspectives of current C^2 system software provide sufficient reason to pursue both better software and better ways to develop it: The historical view shows that more is expected of C^2 systems than the current technology can provide; the research view points out what are probably the most common C^2 problems and offers a framework for trying to solve them; the leadership view gives a vision and time-frame for implementing the systems that commanders need; and, the financial view gives a dose of fiscal reality and some measures to take to reduce the costs for new systems.

B. PURPOSE OF THIS THESIS

The specific purpose of this thesis is to investigate common two-dimensional map and symbol manipulation functions used in current C^2 workstations and to provide a reusable and portable user interface that allows some of the common functions to be implemented. The reason for studying common map and symbol manipulation functions is that a core component of many C^2 systems is the one that displays cartographic and track information. Identification of the common functions and developing reusable software to implement them is one way to help reduce the cost of developing similar software in the future.

1. Primary Goal

The main goal of this thesis was to identify common and essential software functions for manipulating two-dimensional digital maps and associated symbols, and to describe a reusable design of the function software. The primary sources of the functions were the user and design documentation for eight command and control workstation systems. These sources described systems from several diverse operational, developmental, and research areas. The application areas range from atmospheric defense to wargame simulations. The map and symbol functions were designed and partially implemented as reusable software components.

2. Additional Goals

A secondary goal of this research was to investigate the maturity level of Ada programming tools for developing interactive map functions for command and control systems.

A key question was whether or not Ada tools and libraries are mature enough to implement these types of systems.

Another secondary goal was to identify sources and formats of two-dimensional map data for command and control systems. This thesis discusses the digital map data from sources such as the Defense Mapping Agency. Because different formats of data are suited for different types of applications, the main focus was on whether or not the particular data is a good candidate to be used with interactive command and control software.

C. SCOPE

This thesis concentrates only on common two-dimensional map and symbol manipulation functions used in command and control workstation applications. The application areas of focus are evaluation (situation monitoring) and decision-making tools.

The map manipulation functions in this thesis are described from a user oriented point of view. More comprehensive studies have focused on breaking these functions into very detailed sub-functions [64]. Because this thesis is about a relatively small number of common functions, the detailed breakdown is not included except as shown in the software design.

This thesis also considered additional C^2-related functions and user interface characteristics that an operator may desire but which are not necessarily described in the source documentation. However, an exhaustive analysis of potentially useful functions was not part of this effort. The reusable software implementation was restricted to a subset of the common functions.

The discussion of the maturity and availability of Ada software development methods focuses on workstation capabilities in regards to the hardware and system software, software lifecycle support, and other support software issues.

Finally, the discussion on map formats and sources concentrates on map representation issues such as accuracy, storage requirements, and on the availability of the different formats. Although most of the map discussion centers around 2D data formats, some 3D issues are considered as well.

D. RESEARCH APPROACH

This thesis was conducted in three parts. First, a literature search and two site visits were done to find sources of common map manipulation and associated functions. Emphasis was placed on sources that described implemented systems or those currently in development. The same was done concerning software reusability issues and map data sources. Chapter II describes the map functions, development support issues, the maturity of Ada-related tools, and map sources and formats.

Secondly, selected methods were studied and tools learned for use in analyzing and designing the functions. Chapter III shows the results of the analysis and design.

Finally, a subset of the common functions was implemented within a prototype command and control application loosely based on an atmospheric defense system. The software components are documented to enhance reusability. In addition a guide is provided for programmers who may want to use the software as a starting point for future applications.

II. CONCEPT EXPLORATION

A. COMMON MAP MANIPULATION FUNCTIONS

For this thesis, identifying common map and symbol functions in C^2 applications mainly involved comparing the source documents for similarity. Sometimes this required correlating different terminologies and techniques. For example there is more than one way to describe and implement the action of "zooming" a map from a user's point of view. One application used the term scaling instead of zooming. Also, a user could invoke the function from a menu or by selecting an icon. Despite these differences, the functionality was very similar. This section first describes the sources of the common map and symbol functions, and then describes the functions themselves.

1. Sources of Functions

The common functions in this study came from eight sources. The source documentation describes command and control systems that are in use or under development. All of the systems use two dimensional maps. One system also uses three dimensional maps and is included to help illustrate the future direction of command and control system map displays.

a. Command and Control Information System

The first source for map and related symbol functions was a user's manual for the Command and Control Information System (C^2IS) used at the U.S. Central Command Headquarters. The purpose of C^2IS is to enhance the commander's "capabilities for tactical execution and employment by improving the flow of decision making information..." The system includes real-time battlefield displays, data displays, and support for fully automated briefings. This system is operational and runs on a combination of commercial workstations and personal computers. The workstations handle the real-time map displays. [66]

Figure 6 shows an overview of how C^2IS processes information. The situation display segment contains the map and symbol (element) functionality. In this paper, overlays are considered as special kinds of maps that can be edited by the user. However, functions specifically related to overlays, such as editing, are not discussed in this thesis.

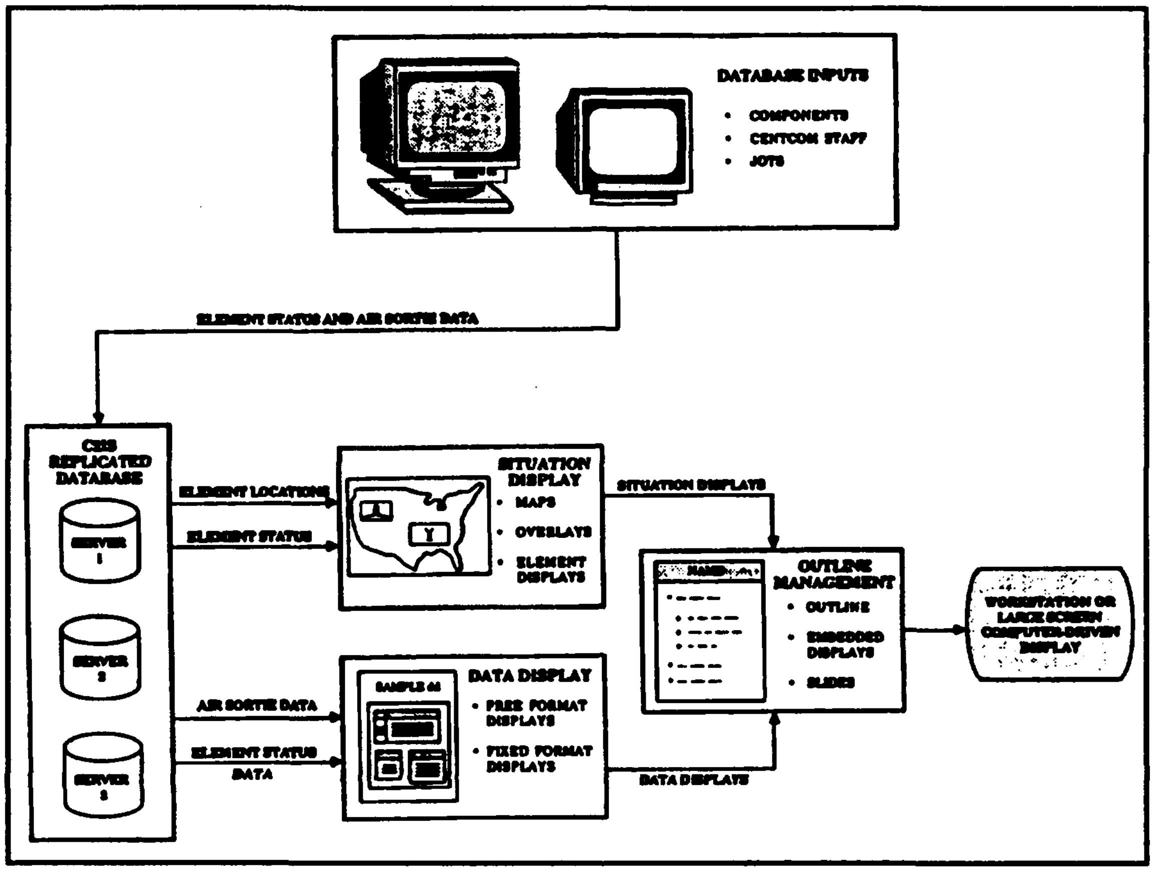


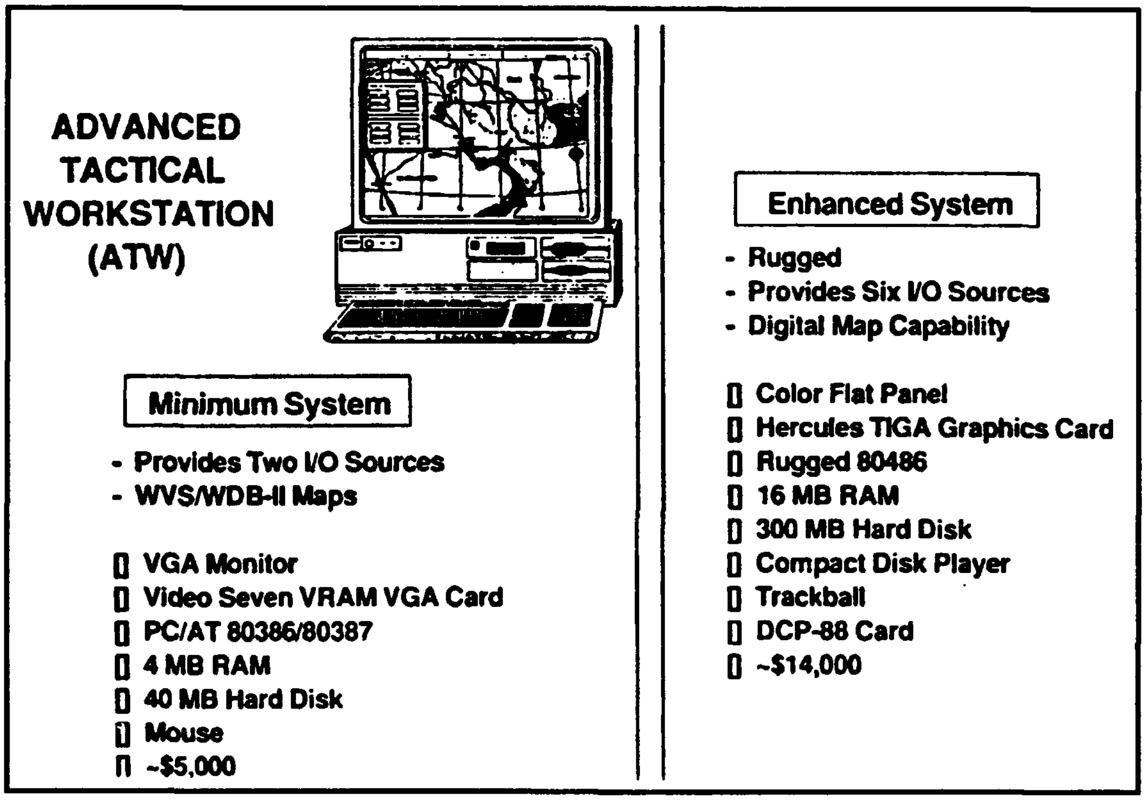
Figure 6. C^{2}IS Processing Overview [66]

b. The Joint Theater Level Simulation System

The second source was a graphics user guide for the Joint Theater Level Simulation (JTLS) system. The JTLS is a highly interactive, computer-based war game system sponsored by the Joint Staff. It provides a tool for theater commanders to plan and simulate the execution of theater-wide operations. Of all the sources discussed in this paper, JTLS probably has the most advanced user interface. It uses networked workstations and video monitors to concurrently display several different types of maps and information. The system can display information in a very timely manner. [22]

c. *The Advanced Tactical Workstation*

A users' manual for the Advanced Tactical Workstation (ATW) from Tiburon Systems (a commercial vendor specializing in C^2 systems) was the third source for functions. The ATW "is a prototype of next generation Navy tactical data processor workstation functionality." Although it is based on low-cost PC technology, it has some impressive features. Figure 7 shows two common configurations. Its greatest strength is its use of sophisticated



ADVANCED TACTICAL WORKSTATION (ATW)

Minimum System

- Provides Two I/O Sources
- WVS/WDB-II Maps

□ VGA Monitor
□ Video Seven VRAM VGA Card
□ PC/AT 80386/80387
□ 4 MB RAM
□ 40 MB Hard Disk
□ Mouse
□ ~$5,000

Enhanced System

- Rugged
- Provides Six I/O Sources
- Digital Map Capability

□ Color Flat Panel
□ Hercules TIGA Graphics Card
□ Rugged 80486
□ 16 MB RAM
□ 300 MB Hard Disk
□ Compact Disk Player
□ Trackball
□ DCP-88 Card
□ ~$14,000

Figure 7. Advanced Tactical Workstation Configurations [58]

correlation and tracking algorithms. These formulas combine track data and, if necessary, make inferences about the data to produce a highly accurate picture of what is being tracked, where the track has been, and where it is going. The ATW plots these tracks on digital maps in a near real-time manner. [58]

The ATW is a flexible, real-time system that has been tested in several different types of scenarios. For example, an ATW test system was installed on a B-52 bomber and used to enhance the Joint Over-The-Horizon Targeting (JOTH-T) and threat recognition and reporting exercise. The ATW's extensive communications ability makes it suitable for this and other widely dispersed surveillance command and control systems. Figure 8 shows the major parts of the JOTH-T ATW configuration and where the map display fits into the system. [59]

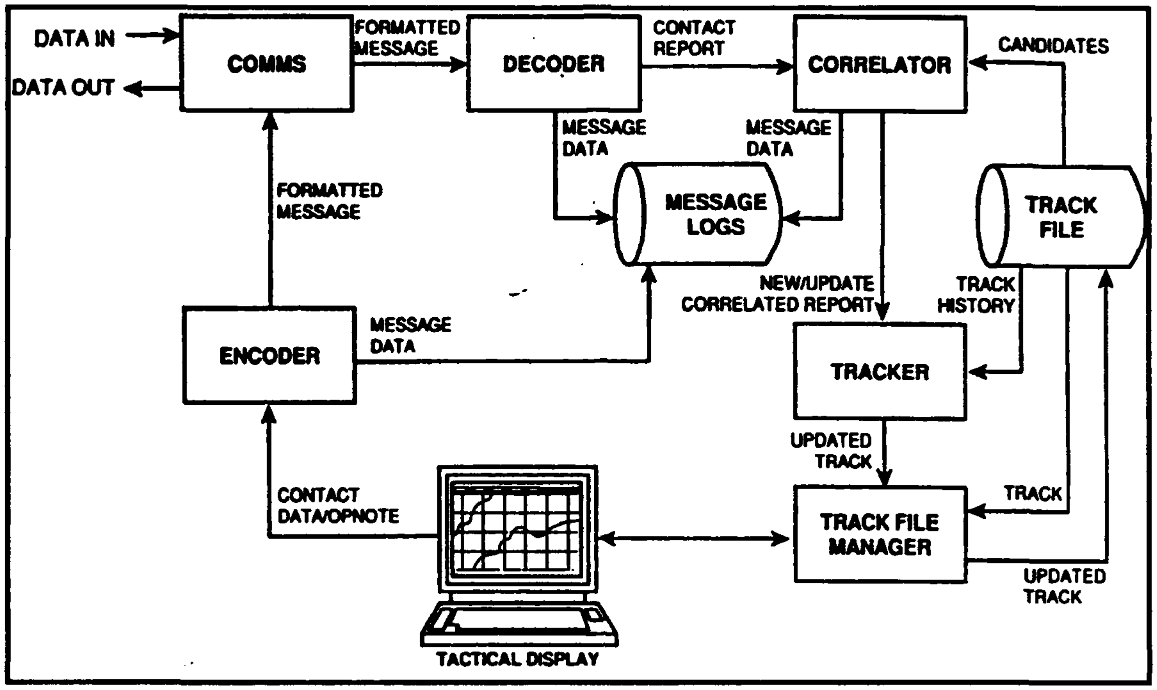


Figure 8. JOTH-T System Data Flow [59]

d. Granite Sentry

The atmospheric defense portion of the Air Force Space Command's Granite Sentry program was the fourth source for map and symbol functions. The whole Granite Sentry program directs the upgrading of the missile warning, space surveillance, atmospheric defense, and intelligence systems in the NORAD Cheyenne Mountain Complex. Figure 9 shows the user interface of the atmospheric defense display. The empty, labeled areas are not shown to keep the display unclassified.

The system configuration consists of networked workstations. The software provides user-configurable, real-time map and data displays which show the mission areas of interest. Any mission display may be shown on any workstation. This allows the decision makers to easily view all of the mission areas. This flexibility also allows the system to be expanded quickly if more operators are needed.[63]

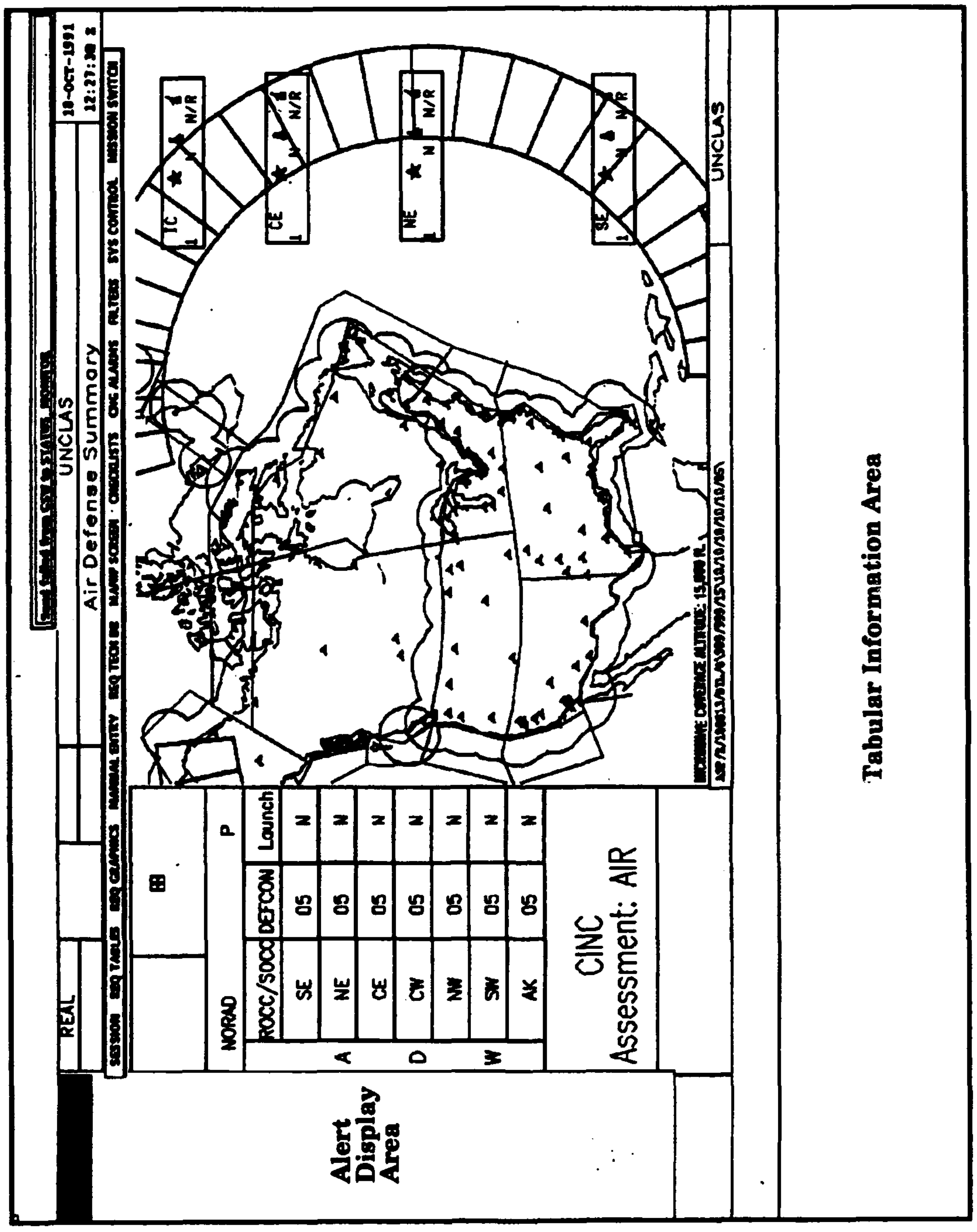


Figure 9. Granite Sentry Atmospheric Defense Display [53]

e. *A Mapping and Graphic Information System*

The fifth source described the concept of a Mapping and Graphic Information Capability (MAGIC) system that World Wide Military Command and Control System users will have in the future. The projected system builds upon capabilities of a current system called the Graphic Information Presentation System (GIPSY).

The MAGIC system will provide a wide range of map and symbol manipulation capabilities. Although MAGIC is not intended for real-time applications, the required quick response time for the mapping functions makes the system attractive for adaptation to real-time use. In any regard, MAGIC will include, in one system, a combination of impressive map and symbol manipulation features not frequently found anywhere else. [30]

f. *Theater War Exercise Graphics*

The graphics system for the Theater War Exercise (TWX) was the sixth source for functions. It provides a graphical user interface for a simulation system run by the Air Force Wargame Center. The graphics system adds mapping capabilities to a previously text-based wargame. An important feature is that the system was designed to be usable, with some modification, for other wargames. [44]

g. *The Multi-Source Integrated Viewing System*

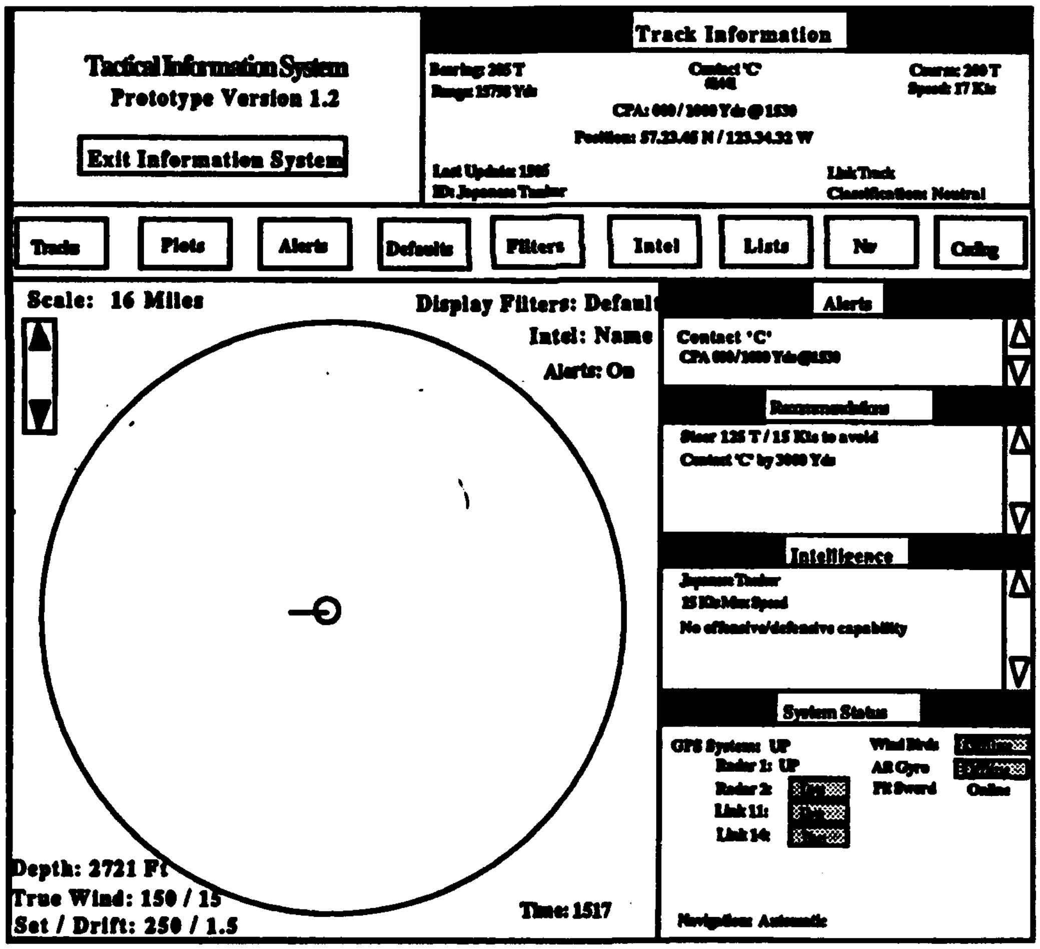
The seventh source was a system the Air Force's Rome Laboratory has produced called the Multi-Source Integrated Viewing System (MIVS). This system provides software tools for digital image manipulation for small computer architectures. This source incorporates functions for both two-dimensional and three-dimensional digital maps. The software is designed to support unit intelligence functions and to be portable to other application areas under various contracts.

This design document source listed several different ways to use MIVS. They include basic cartographic function use, targeting and weaponeering refinement, and damage assessment. In general MIVS provides powerful capabilities for manipulating digital map data of several different formats. [65]

h. The Low Cost Combat Direction System

The last source was project documentation for a research system called the Low Cost Combat Direction System (LCCDS) which is sponsored by the Naval Sea Systems Command. The system's requirements analysis document explained that the sponsors want to install these systems on-board non-combatant ships where no automated combat information processing capability currently exists. This system is to handle the tactical navigation, contact management, and intelligence tasks that are currently being done manually.

LCCDS features are to include the ability to display and manipulate contact symbols, shoreline maps, and overlays. These features and others must be workable on low-cost workstations [31]. The user interface shown in Figure 10 is an example of how the LCCDS display may eventually look [56].


Tactical Information System
Prototype Version 1.2

Track Information
Bearing: 205 T
Range: 15750 Yds
Contact 'C'
[illegible]
Course: 200 T
Speed: 17 Kts
CPA: 000 / 3000 Yds @ 1530
Position: 57.23.46 N / 123.34.32 W
Last Update: 1505
ID: Japanese Tanker
Link Track
Classification: Neutral

Scale: 16 Miles
Display Filters: Default
Intel: Name
Alerts: On

Alerts
Contact 'C'
CPA 000/3000 Yds @1530

Recommendations
Steer 125 T / 15 Kts to avoid
Contact 'C' by 3000 Yds

Intelligence
Japanese Tanker
15 Kts Max Speed
No offensive/defensive capability

System Status
GPS System: UP
Radar 1: UP
Radar 2: [illegible]
Link 11: [illegible]
Link 14: [illegible]
Wind Birds [illegible]
AR Gyro [illegible]
Pit Sword Online
Navigation: Automatic

Depth: 2721 Ft
True Wind: 150 / 15
Set / Drift: 250 / 1.5
Time: 1517

Figure 10. A User Interface for LCCDS [56, p. 50]

2. Map and Symbol Function Descriptions

This section describes the common map manipulation and symbol functions and gives examples of user actions that could invoke each function. The actual user actions used in the software developed for this thesis are described in the software design in Chapter III. In addition, this section mentions some additional functions and features common to command and control applications and lists some user interface issues to consider.

The main functions are divided into two parts. The first part lists the functions dealing directly with the map data. The second part records the functions associated with the map symbols. Some overlap seems to exist. For example, it is not clear if it is best to treat a user-drawn map overlay like a map or a symbol. For this study the decision was made to treat map data separately from anything the user may want to add.

The additional features give an example of what else a command and control map-based application might include. Some of these features could be implemented with operating system and windowing software utilities. For the sake of completeness, suggested user interface features are also listed.

a. *Map Functions*

The map functions are listed in no particular order. User techniques for executing the functions follow the function description.

(1) Zoom Map - simulates moving closer or farther away from the map. Scaling of symbols may or may not be considered depending on the application. One or more of the following interface techniques are used by the source applications:

- a) Enter the scale manually.
- b) Select a preset scale from a list including half scale, double scale, and resetting to the original scale. Scales can be based on altitude or percentages.
- c) Select the next scale (larger or smaller) by clicking an icon.
- d) Select a scale from a list or directly select an icon and then click the pointer on one corner, drag a feedback (outline) box to enclose an area, and click again to anchor the feedback box.

(2) Recenter Map - displays the default area, or an area or symbol selected to be the middle of the map display area.

- a) Select the function from a menu and click the cursor on an area or on a symbol to be recentered.
- b) Select a glyph (basically a movable icon) and put on the center point.

c) Enter the latitude and longitude manually.

(3) Change Map - displays a different map either having different characteristics, showing a different area, or both.

a) Select the function from a predefined map list.
b) Select function and enter coordinates manually to get map of interest.
c) Select different projection of the same map.

(4) Scroll Map - moves the map in the selected direction.

a) Select a direction arrow. All symbols move along with the map.
b) Select a moving symbol as a reference point with which the map tracks (also called panning which means the view of the map looks like a camera is following the symbol).

(5) Show Location - displays the latitude and longitude of the selected location.

a) Select the function using a menu or an icon and click on the location to show the latitude and longitude in a fixed area on the screen or next to the cursor. Coordinate display may vanish or be shown constantly and change if moving symbol was selected.
b) Displays location automatically when a symbol is selected.

(6) Calculate Distance - calculates and displays the great circle distance between two points assuming the Earth is totally smooth and perfectly round.

a) Select the function using a menu or icon and then select two points. Shows a distance value and a line between the two points. Value may be displayed next to line temporarily or in an information list. It could be invoked when calculating a range and/or heading between a reference point and another location.

b. *Symbol Functions*

(1) Show Symbol Information - displays information about a selected symbol or group of symbols in an appropriate format.

a) Displays a list, table, legend, and/or icon depending on the symbol selected. Symbols may include military units, targets, operation sectors, etc. Symbol information could include:

- long name
- short name
- position
- posture
- strength
- radius or weapon range
- related symbols
- speed
- direction vector
- category (friendly, foe, neutral, unknown)
- combat status

- size
- id number
- range and bearing
- estimated time to point of interest
- height (altitude/depth)
- closest point of approach (CPA)
- information source

(2) Edit Symbol - allows user to add, change, or delete map symbology that is application created or user-drawn.

a) Invoke a drawing tool that creates figures compatible with the application. Drawing capabilities could include:

- lines
- rectangles and squares
- ellipse and circles
- polygons
- sketching
- entering coordinates
- icons
- points
- arcs
- text

(3) Display Symbol- displays any symbol such as a track, grid, set of map features, overlays. Depending on the implementation, the symbol can be automatically or manually aligned to specific map coordinates and/or features.

a) Select from predefined list of icons.

b) Display a user-created symbol.

c) Enter boolean expressions or select symbol categories from a menu to display certain symbols.

(4) Remove Symbol - temporarily removes a symbol or group of symbols from the map display. The selected symbols are not permanently deleted from the system.

a) Select the function from a menu or icon and then select the symbol(s) to remove.

b) Enter boolean expressions or select symbol categories from a menu to remove certain symbols.

(5) Other Symbol Functions - the following function was not considered common but was found in some of the sources and was determined to be useful.

a) Calculate Closest Point of Approach: determines the closest distance two selected symbols will attain based on current position headings.

c. *Function Summary*

TABLE 1 shows the common map functions that each source implements. An "X" means the source system included that function. As would be expected an older simulation interface like TWX has fewer features than a newer, and more flexible, system like MAGIC.

TABLE 1. COMMON MAP FUNCTIONS

Sources & Functions	a. C^2IS	b. JTLS	c. ATW	d. Granite	e. MAGIC	f. TWX	g. MIVS	h. LCCDS
1. Zoom	X	X	X	X	X	X	X	X
2. Recenter	X	X	X	X	X		X	X
3. New Map	X	X	X	X	X		X	X
4. Scroll		X	X		X	X	X	X
5. Location	X	X	X	X	X		X	X
6. Distance		X	X	X	X		X	X

TABLE 2 shows the common symbol functions that each source implemented. An "X" means the source system includes that function.

TABLE 2. COMMON SYMBOL FUNCTIONS

Sources & Functions	a. C^2IS	b. JTLS	c. ATW	d. Granite	e. MAGIC	f. TWX	g. MIVS	h. LCCDS
1. Show Info	X	X	X	X	X	X	X	X
2. Edit	X				X			X
3. Display	X	X			X	X	X	X
4. Remove	X	X			X	X	X	X

A trend for more features and flexibility is apparent for both map and symbol functions in C^2 applications. As might be expected, the newer systems have more capabilities than the older ones.

d. *Related Command and Control Functions*

The following functions are not directly related to the map or symbol functions but provide useful information for C^2 workstation operators.

(1) Show current time and date - shows a standard time such a Zulu time. Many of the sources routinely displayed this information for the user.

(2) Log session data - saves live or exercise data for later use or review.

(3) Show exercise time - shows the elapsed time of an exercise while the application is being used for training. This time could also be related to when the exercise data was created.

(4) Show classification - marks the data sensitivity level of what is on the display. (e.g. Unclassified, Secret, etc.).

(5) Change classification - alters the data sensitivity of the display to make it easier to show visitors a system's capabilities. This function could invoke a filtering function to automatically remove any sensitive information from the display.

(6) Display alert messages - shows messages that pertain to external sensors, or symbol related information. This function was not common but seems to be useful in general.

(7) Send and receive messages - allows the user to send messages to and receive messages from other users at local or remote workstations for the purpose of task coordination or information passing.

e. *Related Screen and Window Functions*

Most of the following functions can be performed by system operating system utilities or windowing software.

(1) Refresh screen - redraws the whole screen to remove any extraneous information from the display. This seems to be necessary at times on even the most sophisticated workstations.

(2) Print screen - captures an image of the whole screen to a hardcopy printer or a data file. This can be used to keep a history of the state of the display for later viewing.

(3) Print window - captures the image of a specific window to a hardcopy printer or a data file. This can be used to print just the map display or some other window. This function could be implemented together with the print screen function.

(4) Resize window - expands a window to the full size of the display or shrinks a window back to its original size.

f. *Related User Interface Features*

(1) The following techniques, if used wisely, can help make a command and control application easier to use:

a) Use buttons, icons, and glyphs to implement function shortcuts. Buttons implemented using text labels or as icons are now very common to find in an application user interface. Glyphs, which are closely related to icons, also seem popular, but a common definition of a glyph was not found. Arguably, icons are usually digital image representations of real-world things. For example, a clock icon shows time like a real clock. Glyphs are like icons except more abstract. An example of a glyph is a question mark used to denote a help feature. In this case a function is first chosen and the mode change is indicated by the pointer image changing shape. This is only slightly different from a pointer image changing shape when it is moved from one window context to another. An example of this feature occurs in most drawing programs when an arrow is shown outside the drawing area and a cross-hairs is shown when the pointer is in the drawing area.

b) Provide context sensitive help.

c) Provide an online manual.

d) Allow stacked commands for redrawing map to save time for complex setups. Even on relatively fast workstations, certain map operations may take a significant amount of time.

e) Cancel redrawing if necessary. It may be necessary to interrupt extensive redrawing to allow a response to an urgent event. This could be user and/ or system controlled.

f) Provide an undo command for the last map or symbol function performed.

g) Provide feedback for functions performed. Display the resulting information in an appropriate location for an appropriate amount of time.

h) Use appropriate pointing (input) devices such as a:
   - mouse
   - graphics tablet
   - trackball
   - light pen
   - finger (touch screen)
   - combination of the above

(2) Other user interface considerations could include:

a) Automatically change the pointer shape based on the context of the window in which it is located, as was mentioned in the previous paragraph.

b) Use function keys to implement frequent commands.

c) Use color combinations wisely.

d) The bigger the map the better. Several sources indicated that it is important to closely consider the trade-off of having a large map viewing area and readily available function mechanisms. The use of a "mini" map was also considered valuable to help quickly identify where a "zoomed-in" main map was focusing.

e) Enhance speed by reducing the number of drawing objects.

f) Pull-right (nested) menus should be avoided because they lead to buried, and often, lost functions.

g) Lead the user through difficult functions. Avoid relying on the documentation.

h) Validate data entries and changes. In addition, leave in place previously entered data in entry fields until changes are needed. This helps avoid data reentry for some kinds of functions.

i) Use a second display terminal for text, if possible. Even with windowing workstations, the display can easily become cluttered and multiple windows cannot be displayed at the same time.
j) Gray-out or eliminate entries not used in menus when in the appropriate context.
k) Use hooking where appropriate. Hooking refers to selecting an area or symbol before performing a function.
l) Assume different kinds of user expertise such as novice, average, and expert. Even when C^2 application users have to be fully trained before actually using the system, graduated levels of help can reduce the learning curve. Techniques for this kind of help can include:
   - a training mode for additional prompting
   - macros
   - programmable function keys

This section has described the common map manipulation and associated symbol functions common to eight C^2 map-based systems. One additional system worth mentioning, even though its late discovery prevented it from being used as a source, is a prototyping tool used at the Rome Laboratory. The Lab has developed a map user interface rapid prototyping tool that has been used for preliminary evaluation of new systems similar to Granite Sentry. A Signal Magazine article reported on a comparison made between the time it took the Granite Sentry developers to create a prototype of the atmospheric defense interface using Ada and the time it would take a development team to create a similar interface using the new prototyping tool. Rome personnel determined that what took about a year to do in Ada would now take about six weeks using the prototyping tool. [45]

In addition to the functions, other typical workstation functions, features, and user interface related items were mentioned to point out other common attributes of a C^2 workstation display. In essence all of these attributes are empirical requirements for common C^2 map-based applications. With these common requirements and the next-to-be described software environment, this thesis aims to create some reusable software to help develop and test future C^2 workstation concepts.

B. COMMAND AND CONTROL SOFTWARE DEVELOPMENT

This section covers software development issues on workstations used for C^2 systems with an emphasis on looking at the maturity of the Ada development environment. The issues include workstation and system software capabilities, choice of development lifecyle methods, and the software development support environment. These topics are discussed in the context of describing the available hardware and systems software, evaluating software development lifecyle methods, and evaluating current tools and libraries used to support software development. Other topics discussed include the software quality factors of portability and reusability. The first half of this section discusses the issues in general while the second half explains how Ada relates to them specifically. Chapter III will address these issues in light of the chosen software development methods.

1. The General Software Environment

The current software development trend for workstations is towards open systems and distributed computing which are helping to make more products and processing power available at every stage of the development lifecycle. However, a word of caution is given by Ed Yourdon concerning the choice of new products and tools for development. He warns that focusing too much on one promising technology, especially a new programming language, can lead to developing a "brilliant solution to the wrong problem." He also says that to consider a programming language as the key to solving software problems may "help you arrive at a disaster sooner that before." [69, p. 27] No matter the kind of software development, C^2 applications included, careful consideration of the products and tools for the whole lifecycle is needed.

This part of the paper describes the current capabilities of relatively low-cost workstations and systems software. In addition, software development lifecycle methods and associated tools are discussed as well as different kinds of support software. Particular attention is paid to areas that are important for developing map-based C^2 systems.

a. Workstation Hardware

It would be ideal to be able to first specify and design a C^2 system without any regard to the computer platform. However, in reality, capabilities of the hardware and systems software must be considered.

Low-cost workstations can now execute 20-30 million instructions per second which means advanced capabilities such as windowing system software and animated color graphics can be supported. It was only a few years ago that much more expensive machines were needed. According to a 1988 Naval Postgraduate School report, desired features for a C^2 workstation of the future included color displays, multiple windows, three-dimensional (3D) graphics, networking capabilities, and significant data storage space [38, pp. 2-6].All of these features can be found in current workstations and for a reasonable price. However, it is still true that realistic-looking 3D graphics require relatively expensive, super-workstations to implement real-time scenes in multiple windows [35].

Although 3D graphics are considered beneficial for C^2 systems, currently they are not widely used operationally. Many C^2 systems incorporating low-cost workstations have implemented 2D color graphics in multiple windows. The color displays allow maps, symbols, and text to convey more information more effectively and with less clutter than those without color. The ability to display graphics in multiple windows helps the user quickly view information in a variety of different ways and to accomplish several tasks at a time.

Workstation networks provide access to external information and allow multiple machines to handle the workload. Current networking hardware and protocol software can handle many data traffic requirements. However, some applications could use more bandwidth. Fortunately, advances in networking hardware will continue to reduce system response times and increase processing capabilities.

Data for maps and symbols requires a large amount of disk storage. Relatively fast access times are needed to allow quick updating of information onto the screen. If local workstation storage is not adequate, network access to other disks can be used. However, moving large amounts of digital map data over a network is one of those applications that could use more bandwidth than is currently available.

For C^2 systems it would be useful to have several processes running at the same time to handle simultaneous activities. However, current low-cost workstations do not have multiple processors to allow concurrent processing. The current uniprocessor systems simulate concurrent processing through system software features. But it is probably just a matter of time before multiprocessor systems become common.

b. Systems Software

For a generic C^2 workstation, special system software capabilities should be provided. The system software should be able to satisfy certain timing constraints. Also, it should also allow the application software to run on different kinds of workstations. In other words the operating system should support real-time information processing and display, support multiple windows, and support standard application programming interfaces (API) so software can be easily used across different systems.[9]

(1) Real-Time Considerations

Many C^2 applications require the system to respond to important events as they happen. This kind of processing is usually called "real-time" and often involves external interfaces. For example, an atmospheric defense system has to accurately track aircraft and classify them according to their potential threat to friendly forces. Failure to do this in a timely manner can be costly.

System software to support real-time constraints does exist, but it is not common on workstations. The most common operating system (OS) for workstations is usually a version of UNIX. This OS does not inherently support real-time processing. However, extended and less portable versions of UNIX that support real-time applications have been developed.

(2) Window Software

For multiple window support and a consistent user interface, windowing software is necessary. This kind of software provides a facility for developers to create applications that can be executed on different types of workstations in a stand-alone or network configuration. For example, on Sun workstations the most common windowing system software are SunView and OpenWindows from Sun Microsystems running with the X Window System software from the Massachusetts Institute of Technology (MIT) [43]. The X Window System (commonly called X) was developed primarily at MIT and is available for many different workstations.Vendors are also embedding X software into terminals to provide a lower cost windowing display device. The current version of X is called version 11, release 5 (X11R5). However, most vendors currently support X11R4.

In addition to X, there needs to be software that provides a consistent user interface. X only provides a protocol to display windows on a screen, unlike SunView which specifies both the protocol and look-and-feel. X does not specify how the windows should look to the user. The user interface software usually provides a window manager and a set of programming tools so a developer can add the look-and-feel to custom applications. Two of the most common user interfaces are Open Software Foundation Motif (OSF/Motif) and UNIX Systems Labs OPEN LOOK, on which OpenWindows is based. Each product comes with a window manager, a programming tool set, and implementation guidelines. Each product provides similar functionality and has its own advantages and disadvantages so the user is left to decide which one to use.

Most of the C^2 map function source systems use some sort of windowing software. For instance, the C^2IS system uses SunView. The Granite Sentry system uses a version of X called DECWindows, a Motif compliant interface, was developed by Digital Equipment Corporation (DEC), one of the corporate sponsors of the MIT X development.

(3) Standard Application Programming Interfaces (API)

The goal of standard APIs is to provide mechanisms for software portability and reuse. The primary API standard between the operating system and application software is called the Portable Operating System Interface (POSIX). Although POSIX is not yet an official standard, many vendors do provide systems software that meets the specifications of the draft version. The official version of POSIX should provide a standardized way for applications to interact with system resources.[19]

An internationally standardized API and graphics system called the Programmers Hierarchical Interactive Graphics System (PHIGS) is gaining acceptance. This system allows a software developer to define, store, and display 2D and 3D graphics models using portable terminology within the application [25]. A PHIGS Extension to X (PEX) is supported in X11R5.

The future for systems software seems headed toward distributed, concurrent processing operating systems. With their frequent need for parallel operations, C^2 systems should benefit from multiprocessor systems and software that can utilize them.

c. *Lifecycle Support*

Building sontware for C^2 workstations requires more than just a machine and the system software. The developer needs to have some development tools and guidelines to successfully map the system requirements into software. Developing C^2 software is not much different from other large software developments. Therefore, similar steps and techniques can be used.

This section describes some common methods and tools used in the first three phases of the software development lifecycle: requirement analysis, design, and coding. The discussion of the phases is followed by considerations related specifically to C^2 software.

(1) Requirements Analysis

The first phase in most software development lifecycles is requirements analysis. According to Pressman, "requirements analysis is a software engineering task that bridges the gap between computer system engineering and software design." [42, p. 174] He goes on to mention issues an analyst should consider in this phase such as software function, performance, and interfaces to other systems.

There is no one sure way to successfully analyze software requirements. Several methods are considered useful including structured analysis and, more recently, object-oriented analysis. Other less formal techniques are also considered useful in this phase. Several kinds of tools are available for doing analysis using these methods and techniques.

The structured analysis method is probably the most widely used method. It basically involves looking at the requirements in terms of data and functions. For this method there are quite a number of Computer-Aided Software Engineering (CASE) tools that give the analyst the means to draw a model of the system in a graphical form. An example of such a tool is Software Through Pictures (StP) [26]. StP allows the analyst to create several kinds of modeling diagrams and store them for use in the design phase that follows.

The object-oriented analysis (OOA) model is the most recent paradigm advocated for requirements analysis. Proponents argue that thinking in an object-oriented manner is more intuitive than using other techniques. They also say using OOA helps reduce the "cognitive leap" that a developer has to make to the associated object-oriented design and object-oriented programming methods which are becoming widely used. [50]

Currently there is no agreed upon method of employing OOA. In general OOA involves thinking of the problem area in terms of things (objects) and categorizing them using some taxonomy. Currently not many tools support graphical diagrams for documenting OOA concepts.

Software prototyping is considered a less formal technique for analyzing software requirements and is often used in conjunction with other methods. Software prototyping is a process of creating, testing, and refining a working or paper prototype of what the software should do. Automated techniques are usually preferred since they allow realistic interaction and the ability to quickly make changes. Pressman explains that requirements analysis prototyping methods and tools include fourth-generation techniques (4GTs) and/or reusable software components. [42, pp. 192-193]

An example of a tool using 4GTs is the Transportable Application Environment Plus (TAE Plus) software development environment for user interfaces developed by NASA [61]. With this kind of tool, an analyst can build interactive interface display models to match the requirements so the user has something substantial to assess. This process can cross over into the design phase, but it is valuable in giving the user and analyst something firm on which to base the requirements. Pressman points out that prototyping may have to start at the beginning of analysis "since the model is the only means through which requirements can be effectively derived" [42, pg. 190]. Along with the prototyping tool, reusable software components can be used to add functionality to the prototype to capture even more of the requirements.

(2) Design

Software design is a process by which requirements specifications (created during the analysis phase) are translated into specifications suitable for software implementation. In other words, the designer has to describe the real-world problem in a form that can be easily transformed into software. Pressman says the importance of design to the developed software can be described in one word—quality [42, p. 317].

Currently, the most widely advocated paradigm for software design is object-oriented design (OOD). Pressman describes OOD as a method that "creates a representation of the real-world problem domain and maps it into a solution domain that is software." [42, p. 395] What actually constitutes an OOD method is not universally agreed upon. However, one of the most respected proponents of OOD, Grady Booch, describes OOD as the process of identifying the classes and objects, their semantics, and their static and dynamic inter-relationships. Booch explains the process of identifying and documenting such a design as being iterative with the experience of the developer being an important factor. [13, pp. 191-196]

CASE tools that help with design are available and are quite useful. For example, a tool called ObjectMaker [34] lets the designer draw Buhr diagrams [14]. It even automatically generates Ada or C source code. The design diagrams allow the designer to express relationships among higher level programming constructs.

(3) Coding

Once the design has been completed, the developer has to generate the source code so the software can be compiled into the final program. Since many C^2 systems require concurrent and real-time processing, the programming language should support these criteria. The language should also enforce the building of reusable software components so they can be used in future systems.

Not many languages inherently support concurrent and real-time concepts. Ada is one language that does, but it also needs the support of the hardware and system software to implement them fully. On most workstations, concurrency and real-time processing are usually simulated by operating system time-slicing and by schedulers programmed within the application.

Using and creating reusable source code is often a goal in the coding phase. Pressman reports one definition of software reusability as the "extent to which a program (or parts of a program) can be reused in other applications—related to the packaging and scope of the functions that the program performs" [42, p. 552]. Finding applicable code to reuse can be difficult and time-consuming. Likewise, creating and documenting reusable code is a time-intensive process. Structuring the code in a common format and documenting it are the most basic ways to help make code reusable. Recommended coding standards are common for most languages, but they usually have to be enforced by policies. The next section will consider reuse in more depth.

d. *Support Environment*

In addition to the hardware, system software, and lifecycle tools mentioned, there are various kinds of software that can be considered part of a workstation development support environment. This software can include other development tools, configuration management tools, and software libraries to include reusable source code components.

(1) Other Development Tools

Other development tools can range from source code editors to source code pretty printers. The UNIX workstation environment is full of capabilities when it comes to programming tools. Examples include the Emacs editor and a configuration management tool called the Source Code Control System (SCCS).

(2) Software reuse

In addition to tools, reusable software is becoming an important part of software development on workstations as well as on other platforms.

According to Krueger, software reuse can come in several forms. He lists high level languages, application generators, software architectures, source code components, and code scavenging as some of the different categories for software reuse techniques [32, p. 5]. The most common reuse techniques fall into the categories of source code components and code scavenging. Booch defines a component as a "container for expressing abstractions of data structures and algorithms." [12, p. 7] Although terminology does not seem to be consistent in the realm of reuse, a set of components is generally considered to be a software library. Likewise, a set of software components and libraries can be gathered into a software repository which helps software developers find reusable software.

A fair question at this point is whether or not software component reuse is really making a wide-spread difference in actual systems. According to a panel of experts at a major software engineering conference, reuse is beneficial if the whole development process considers this facet when first designing the software. On the panel's scale of 1 to 5 (1 meaning reuse is not helping to 5 meaning reuse is factually beneficial) the participants in general consider the state of reusable software to be a 4 when handled properly. This means that "there is experimental evidence that reuse is producing increases in software quality and productivity." [23, p. 52] This conclusion was reached upon considering several case studies and many personal experiences. From this panel's point of view, reuse is having a positive impact in well-defined application domains. [23]

Software libraries are commonly used for workstation application development. All of the previously mentioned system software—the UNIX OS, X, Motif, PHIGS—have libraries that can be used by a software developer to create custom applications. It is also common to find source code components developed by others for the UNIX workstation environment. Reusing software is something to definitely consider when building C^2 software on workstations.

2. The Ada Software Development Environment

The proliferation of Ada is quite impressive. It is reported there are more original sources of Ada compilers than for the C++ language and that more universities and training schools teach Ada than C++ [3].

This section describes the maturity of a current Ada software development environment on a low-cost workstation. The emphasis was on determining whether or not the Ada environment is adequate for developing 2D map-based C^2 applications. The tools and methods selected for the thesis software development are also mentioned.

a. *Hardware and Systems Software*

For the software development in this study, the C^2 workstation was a commercially made Sun SPARCStation 2 desktop system with a large screen color display. The system was connected by a local area network to a file server for additional data storage and development tool access.

The system software included the UNIX compatible SunOS 4.1.1 operating system [57] along with X and the OSF/Motif user interface. X and OSF/Motif were chosen (rather than SunView or X and OpenWindows) since they were readily available and development tools such as TAE Plus required them.

As for using Ada on this kind of system, many vendors make compilers and development tools that work specifically for this configuration [40]. The Sun workstation was able to provide impressive response times when running the OS, X, the Motif window manager, and Ada applications.

b. Lifecycle Support

This section describes the roles Ada and compatible tools can play in each of the previously mentioned software development phases.

(1) Requirements Analysis

For this project, prototyping was the primary method used to specify the requirements. TAE Plus was the tool of choice. A specific benefit of TAE Plus is that it can generate Ada code which can later be modified to provide application-specific functionality. Prototypes used to be considered mostly "throw-away" code; but with tools like TAE Plus, much of the analysis effort can be used for the software implementation.

Using reusable software components to aid prototyping is a way Ada can help during the requirements phase. If a set of existing Ada components exists and the right one can be found easily, the analyst can build a working prototype much more quickly. Together with a tool like TAE Plus, these components can provide the functionality missing from the generated code.

For this study, the function requirements research and initial prototypes created with TAE Plus were essentially the methods used for requirements analysis. Ada played a role by allowing the prototype to be compiled and run independently for faster execution. Chapter III covers the analysis in more detail.

(2) Design

In the design phase Ada can play a significant role depending on the method used. Booch advocates using Ada as a program design language for parts of the design when using the object-oriented method [13, pg. 157]. It is important to keep in mind Ada does not fully support concepts used in the object-oriented paradigm, although Ada 9X should make this more straight-forward in the future [2]. In his book, *System Design with Ada*, Buhr offers an "object structured design" diagramming technique which maps directly to Ada concepts [14].

CASE tools that help with design and that generate Ada code are available and are useful. As previously mentioned the ObjectMaker tool lets the designer draw Buhr diagrams and then create Ada source code automatically from the diagrams.

ObjectMaker also allows a designer to take Ada source code, such as that created by a tool like TAE Plus, and generate diagrams from it (this is called reverse engineering). This way the design of the prototype software is available to be modified. A designer can add more functionality to the prototype code diagrams and turn the diagrams back into source code again. This allows the designer to make a complete design diagram and to salvage the work that was put into creating the prototype.

For the software developed in this thesis, Buhr diagrams were generated with ObjectMaker using the source code generated by TAE Plus. The diagrams were then modified to document and add some functionality to the map and symbol source code. Since Buhr diagrams were used, the software design method is probably best characterized as object-structured. However, a previously created OOD for map symbols was used to guide the implementation of air tracks [21]. In this case the design document was a set of class definitions which were added directly to the TAE Plus Ada code before the Buhr diagrams were created. This is described in more detail in Chapter III.

(3) Coding

Ada inherently supports several features that many C^2 systems require. For concurrent and real-time processing, Ada has the tasking and run-time executive features. However, a programmer must implement these capabilities with care. Depending on the programmer's experience, compiler maturity, and underlying hardware, problems can occur. The Ada language itself is not without its own shortcomings. A GAO report on Ada outlines the real-time Ada issues and suggests some solutions. [67]

Ada is no stranger to standards relating to portability and reusability. An important characteristic of Ada is that it is an international (ISO/8652-1987), national and military (ANSI/MIL-STD-1815A-1983) standard. This means the language is strictly controlled and an Ada compiler must pass a battery of tests to be considered validated. This gives a high degree of confidence to developers that software written in Ada can be easily recompiled on different systems.

The compiler technology for Ada is very good. Currently 28 different companies in the U.S. sell validated Ada compilers. Most of them are very mature [3], although some language compiler efficiency issues still exist when implementing real-time systems. However, many experts think these issues should not prevent Ada from being used in most systems. [67, pp. 83-84]

The Ada software for this thesis was developed using the Verdix Ada Development System (VADS) [68]. VADS is probably one of the most mature Ada development systems available [40].

To help Ada programmers achieve reusability, the Software Productivity Consortium has published guidelines on coding standards and reuse techniques [1]. This should help software developers write readable and well documented components.

In addition to Ada, an object-oriented Ada language extension called Classic-Ada was used to implement the OOD for the air track portion of the software. Classic-Ada adds some additional keywords to the Ada language to help express object-oriented concepts. The Classic-Ada source code was translated into regular Ada source code by a preprocessor [51]. The Ada code was then compiled normally. Combining Classic-Ada with Ada was fairly easy to do. The reusability of the software was not greatly affected by using Classic-Ada. This will be explained more fully in Chapter III.

c. *Support Environment*

The workstation software support environment for Ada includes development tools, software libraries, language bindings, and reusable software components.

(1) Other Development Tools

The previously mentioned tools, TAE Plus, ObjectMaker, and VADS provide some of the most significant Ada development capabilities currently available. However, other tools are needed to fill other development needs. For instance, a source code editor is necessary. For this project the Emacs editor was used. Although a language sensitive Ada-mode feature was available, it was not found very useful and therefore rarely used. The Ada-mode did help prevent mistakes but the overhead of entering extra commands seemed to slow down development.

Keeping track of source code software versions is another area where tools can help. The size of this project did not require a configuration management tool, but one might be useful if much more source code is ever added. Keeping track of versions did become a little confusing at times.

Formatting source code to be more readable is a another task where tools can help. The VADS product comes with a pretty printer that was used to make the source code consistent in style. Since TAE Plus, ObjectMaker, and programmer generated code were combined, a pretty printer was used to convert the multiple styles into one.

(2) Software Reuse

Reusing software is becoming an important part of Ada development support environments. Software libraries help facilitate reuse and are widely used on workstations. Unfortunately, most of the system software libraries for UNIX workstations are written in the C language. To use them in an Ada application, language bindings are necessary to map Ada subprogram call formats to the C function call formats. The Ada Information Clearinghouse has a list of many different libraries that have Ada bindings [6]. For this project the Software Technology for Adaptable Reliable Systems (STARS) Ada bindings were used to call X library (Xlib) routines [48]. Also, a small set of bindings were developed to call operating system utility programs and other useful, non-Ada functions.

In addition to using software libraries, some of the Booch components were used in this project to provide calendar functions and data structure operations [12]. They proved to be very useful by reducing the amount of work needed to implement commonly needed routines.

To help Ada software developers find software to reuse, several significant reusable software repository efforts have been started. The original Ada Software Repository (ASR), which is still operating, was probably the first large-scale system to provide an Ada library of reusable software components. [15, p. 78] The ASR contains directories of software source code which cover a range of applications. However, difficulties with this system include a lack of quality control and a lack of browsing features—shortcomings common to most current repositories.

Other agencies have established their own reuse programs. The National Aeronautics and Space Administration (NASA) has the Computer Software Management and Information Center (COSMIC) [33, p. 67]. The DOD has started an effort called the Asset Source for Software Engineering Technology (ASSET) which will develop a comprehensive reuse repository. ASSET currently includes the software components developed under the DOD STARS program [16]. The key now is to make this information easily available to software developers. Appendix C lists more information about these and other efforts.

Overall, the current generation of desktop workstations, the UNIX operating system, the Ada programming language, and current development tools combine to make an impressive environment for developing C^2 applications. With the direction provided by the reuse initiatives, workstation software development for C^2 applications can only benefit.

d. Ada Development Summary

This section discussed just a few of the methods and tools useful for developing software in Ada. It seems the use of Ada in military software development is very strong and capable. It is well documented. The Ada Joint Program Office keeps a database of current Ada development projects [4]. Many of these projects involve real-time command and control systems. The Advanced Tactical Fighter is an example of a program successfully using Ada for a real-time application. In addition, the Granite Sentry program software, mentioned as a source of map and symbol functions, was written in Ada. In general it does look like Ada is successfully being used to implement significant command and control software and is more than adequate to develop workstation-based map applications.

C. C^2 MAP ISSUES, DATA SOURCES, AND FORMATS

This section lists critical issues for representing cartographic data for C^2 applications and compares map formats commonly used for C^2 systems. Analysis of map formats was based on frequency of use, typical application, and applicability to a standard C^2 environment. Based on this analysis and project constraints, a map format was chosen for implementation of the initial set of components. Also, Appendix C contains more specific information concerning digitized map data sources.

1. Map Representation Issues

Most of the issues and limitations that pertain to digitized maps, are the same ones that pertained to the first cartographer's map.

a. Accuracy

Accuracy of data points is a limitation of digitized maps as it is with hardcopy maps. Manual and satellite collection methods are not perfect. The impact of the margin of error is usually method specific and relatively small. Medium resolution data points are usually considered accurate to within 500 meters [17, p. 33]. Small resolution data points can be accurate to approximately 30 meters [17, p. 13]

b. Scale

Certain representations of data are not well suited to scaling. For example, changing the scale of data scanned in from hardcopy maps may distort the shape. Such data must be represented by several scales of the same map, stored together then scaled at only specified increments. Map data stored in a vector structure can be dynamically recalculated based on an arbitrary display scale.

c. Storage

Some map data formats require considerably more storage than others. A full screen raster image of a typical Sun color display requires over 1MB of storage. A typical vector representation of North America with political boundaries requires only about 13KB. If additional raster images are stored to support incremental scaling, storage and access requirements can be overwhelming.

d. Performance

Depending on data content and resolution, some formats require significantly less time to redraw the map than others. When data points are few, vector representations can be drawn very quickly. However, when data points are many, bitmap graphic images may be generated more quickly. Scaling may take longer with vector forms due to the calculations involved, while raster image scaling is typically done using different resolution images of the same map, thus requiring no calculations.

2. Sources and Formats

Digitized maps for command and control applications can come from a number of sources and in different formats. Probably the best source is the Defense Mapping Agency (DMA). Other sources include the Central Intelligence Agency (CIA), and the United States Geological Service (USGS).

a. Defense Mapping Agency

Probably the best source for digital map data for military applications is the Defense Mapping Agency (DMA). The DMA produces over 20 different formats of data and several seem useful for C^2 applications.

(1) World Vector Shoreline: The data used for many C^2 applications is called World Vector Shoreline (WVS). This data contains shorelines, international boundaries, and country names. The intended applications are "Tomahawk shipboard and shore mission planning, AEGIS command displays, Tactical Flag Command Center displays, and various other display C^2 systems." [18, pg. 24] The WVS data is stored in a format which allows quick access to adjoining data and practically unlimited scaling. Data point accuracy is within 500 meters of the true geographical position. Several of the sources for this study use WVS data. Systems which specifically mention its use are the Advanced Tactical Workstation, Granite Sentry, and the Multi-Source Integrated Viewing System. Some of the other sources probably use WVS, but they could not be verified.

(2) World Mean Elevation Data: Where elevation is important for a command and control system, World Mean Elevation Data provides a coarse resolution with continuous worldwide coverage. It contains the minimum, maximum, and mean terrain elevations. Most of this data originates from more specific data called Digital Terrain Elevation Data (DTED). Since DTED does not yet provide worldwide coverage, other, less accurate sources have been used to provide the needed elevation values [17, pg. 17]. Figure 11 shows the basic format of DTED which is basically a uniform matrix of elevation values. The data point horizontal spacing is approximately 100 meters.

(3) Map Digitized Raster Graphics: For C^2 systems that need more map feature detail, the Equal Arc Second Raster Chart/Map Digitized Raster Graphics (ADRG) products

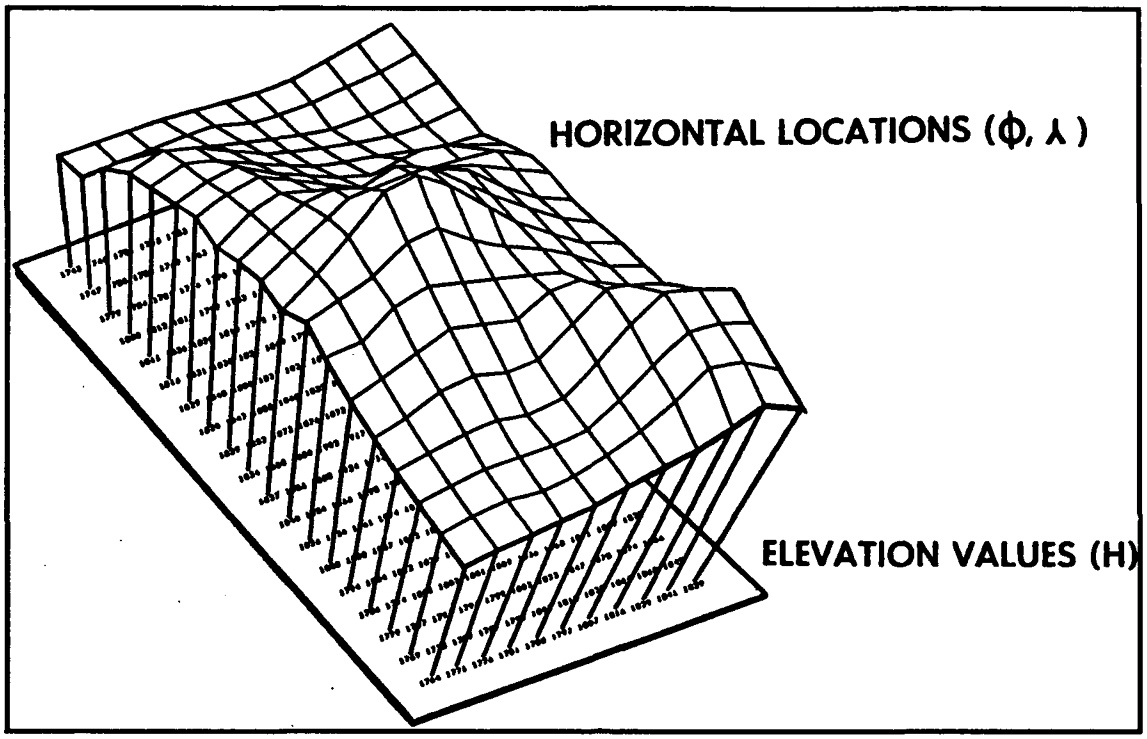


Figure 11. DTED Data Format [18, p. 9]

are available. These are basically digital raster representations of paper graphics products. [17, pg. 41] The Advanced Tactical Workstations (ATW) uses this data by overlaying it onto the WVS data. Since the DMA only produces a limited number of differently scaled ADRG map sets, the ATW uses the WVS map data to allow a map to be displayed no matter what scale is requested. If the ATW shows an ADRG map, the WVS map is hidden.

(4) Map Video Disk: Most workstations take a fair amount of time to display detailed, scanned digital map information. For C^2 systems that need maps quickly redisplayed, the DMA has the Mapping, Charting, and Geodesy Video Disk. It "consists of images of various scale maps and charts stored on a standard analog video laser disc combined with program and database software." It is ready to be used with PC and some DEC VAX hardware. [17, pg. 43] The Joint Theater Level Simulation (JTLS) system displays this data on large-screen TV monitors to compliment the workstation map capabilities.

b. The Central Intelligence Agency

The predecessor to the WVS data are the World Data Bank (WDB) I and II data sets created by the Central Intelligence Agency (CIA). Up until the early 1980s, this WDB data was the primary format for most Navy map-using systems. The main disadvantage to WDB data is its lack of cartographic detail. Figure 12 shows a comparison of WDB and WVS data resolutions. All of the figures show part of the coastline of the Odessa region in the northern

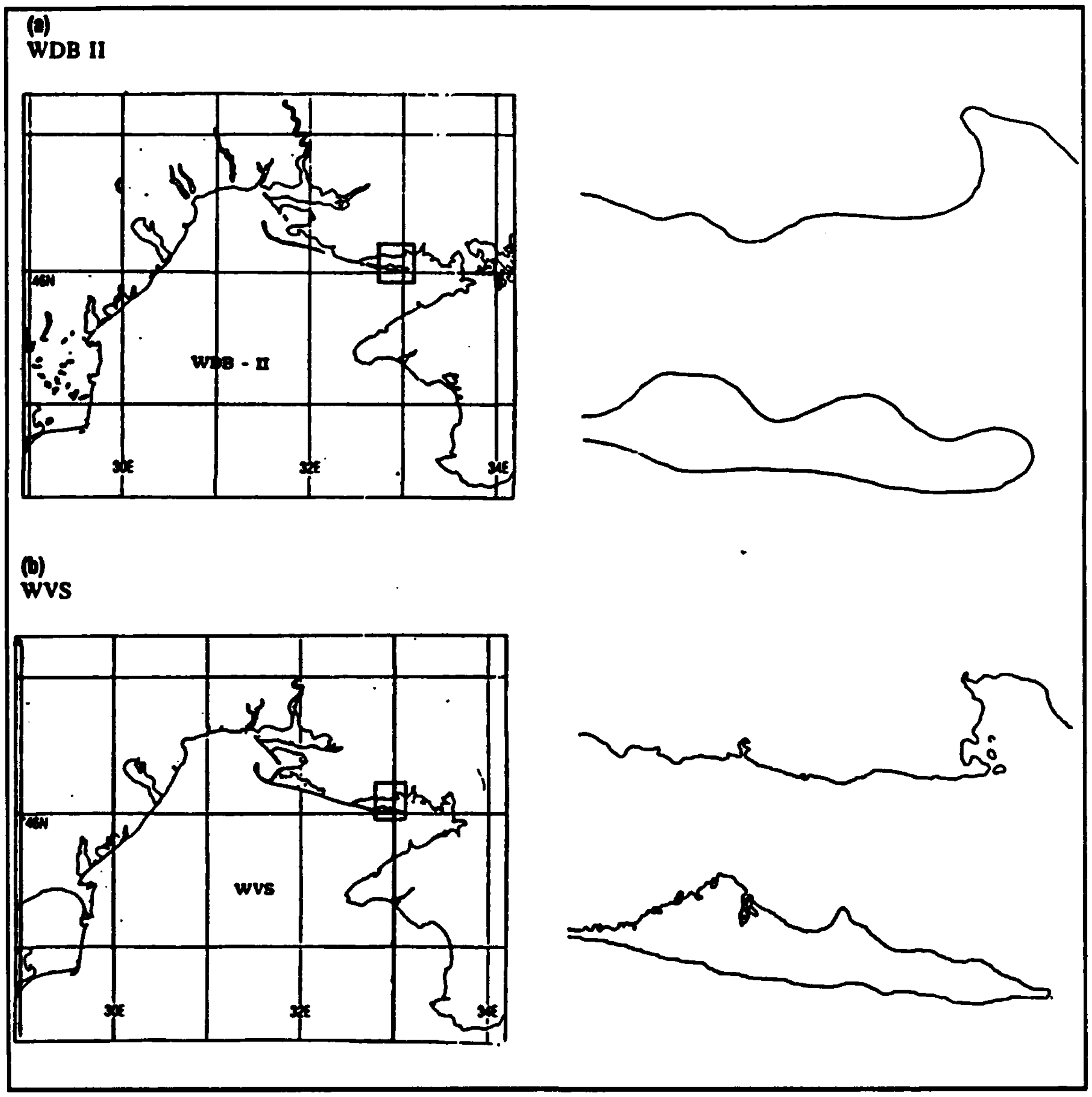


Figure 12. Comparison of WDB II and WVS Data Resolution [37, p. 4]

Black Sea. The top half of the figure (a), shows the WDB II data while the lower half (b) shows the WVS data. The left-side map for each format shows the region at one scale while the right-side map shows the zoomed-in area outlined by the small square inset box in the other map. WDB II was digitized at an average scale of 1:3,000,000 and the WVS data was digitized at and average scale of 1:250,000. [37]

The major advantage of WDB II data is its availability. The whole data set can be obtained via the Internet and software that can manipulate its structure is in the public domain. This is one reason the Theater War Exercise (TWX) Graphics system contains WDB data for its maps. Also, at the time the TWX system was being developed, the WVS data was not generally available from the DMA. [44] Earlier versions of the Advanced Tactical Workstation also used WDB data.

A disadvantage of the WDB II data is that its raw format is not very useful. The data come in three separate files that contain political boundaries, coastlines, and rivers. The coordinates are not labeled with their applicable area names. To make this data more useful, several efforts, both formal and informal, have created derived or expanded data sets that organize the data into more useful formats. One well-supported and recent expanded format is called the Relational WDB II (RWDB2). This format organizes the data and associated names using a relational database. It was originally intended for producing printed small-scale, page-sized maps. However, it is being successfully used in interactive Ada map display software applications. [41] See Appendix C for more information on where to obtain this data.

c. *Other Sources and Formats*

The United States Geological Service has a large amount of map data of the United States. A compact disk (CD-ROM) has been published and is available to the public. The usefulness of this source of data for C^2 applications is not apparent since the DMA has similar data. Its greatest advantage is its availability to the general public. Appendix C describes how to get this data.

Digital image data can be used the same way as scanned image data and is usually collected by aircraft or spacecraft. The Landsat satellites are probably the best known systems for collecting this sort of data. For C^2 these images can be used to gather intelligence which can help the commander make assessments about the enemy's capabilities.[46]

Image data can also be used in the process of synthesizing, or creating 3D maps. By superimposing (or registering) image data from an area onto the related elevation data, a textured 3D map of that area can be formed. As workstations get more powerful, using 3D maps of this kind for C^2 will be more common. Impressive work has already been done in this area. [47, 36]

3. Map Format Selection and Future Trends

Due to the ready availability of raw and derived data as well as sample algorithms for its manipulation, a derivative of the WDB II data was selected for this effort. However, as the design and implementation section of this paper will illustrate, good software engineering techniques for abstraction and information hiding provide a clean mechanism for updating the map format and source in the future. Future components could be developed to implement alternative data formats appropriate for C^2 applications that need them.

The amount of raw map data and uses for it are increasing. The DMA continues to update its digital map products. More and more of it will become available on CD-ROMs. The DMA has several prototype map data types being evaluated.

Besides data and format variety, standards are being developed to control formats and support ease of data exchange. In 1990 the DMA established the Electronic Map Display Interoperability Program in response to a Joint Staff Tasking. One of the program goals is to "assist developers of command and control systems targeted for the Joint Staff, Unified and Specified Commands, and joint activities by guiding development of standard mapping application software." [17, pg. 73] Other standards efforts deal with map data formats and data exchange.

Most of the leading-edge research is now focusing on displaying map data in 3D. Figure 13 shows a glimpse of actual research work. The figure shows a three-dimensional "threat envelope" dome as displayed by the MIVS system. The dome graphically shows the threat area for a given weapon—whether it be a surface-to-air missile or something else.

As the command and control workstation of the future report predicts, more three-dimensional data will prove useful as commanders will need to see even more information displayed in a quickly recognizable format [38].

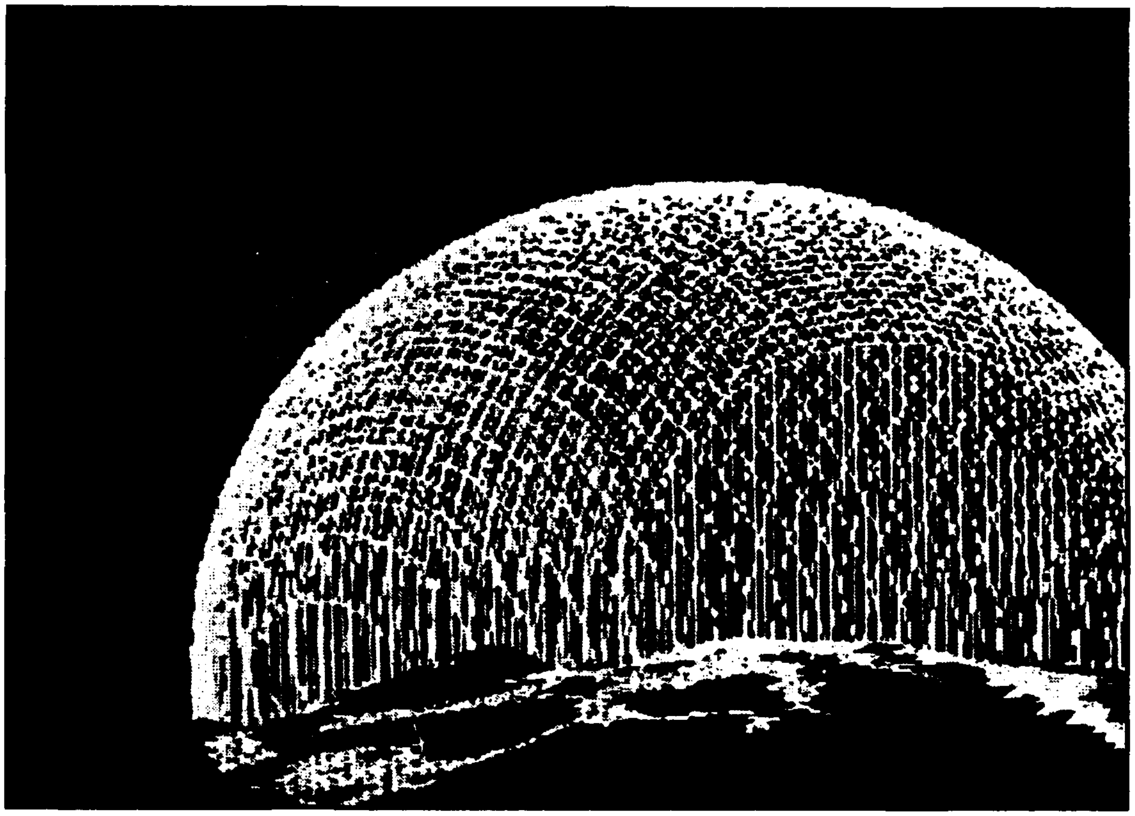


Figure 13. Three Dimensional Threat Envelopes [47 p. 182]

III. SOFTWARE REQUIREMENTS ANALYSIS AND DESIGN

This chapter describes the thesis software development methodology which focuses on software analysis and design methods, tools, and products.

The overall methodology most closely resembles the traditional waterfall software development lifecycle model. In this model the requirements phase is followed by the design phase which is then followed by the implementation, testing, and maintenance phases. This chapter focuses on the first two phases and describes the techniques used and products produced for these phases. The waterfall model also provides for iteratively returning to previous phases when it is necessary to make changes based on discoveries later in the lifecycle. Iteration proved necessary several times for this software development and is mentioned where appropriate.

The users for this project were primarily the author and the thesis advisors. There was also some contribution from human-computer interaction students and the instructor.

A. REQUIREMENTS ANALYSIS

The methods of analysis for the C^2 map and symbol manipulation software included function research and evolutionary storyboard prototyping as described by Pressman [42, p. 192] and Andriole [8]. The function research was described in Chapter II. Based on the common and related functions identified in this research, an interactive prototype was developed and critiqued.

1. Researching the Functions

Almost all of the common functions from the researched sources were incorporated into the prototype and the following software design. All six of the map functions were included. Of the four symbol functions, the editing function was excluded except for the ability to add an external editor. Because of the availability of external editors (such as idraw from Stanford University's InterViews graphics package), and the complexity of writing such software, the decision was made to only provide a way to invoke an editor from the user interface. In fact, this is how the developers of TAE Plus provided some editing features. When using TAE Plus, the way to edit graphics is to use the idraw program or the X bitmap editor—which were created at Stanford and MIT respectfully. [61, p. 241, 235]

In addition to the map and symbol functions, the related command and control, and screen and window functions were considered to be requirements for the software. Many of the user interface features were incorporated where appropriate. Specific features not included were stacked commands, drawing cancellation, alternative input devices (besides a keyboard and a mouse), and a second text display.The rest of the features were used in some way.

2. Evolutionary Storyboard Prototyping

Since software requirements are difficult to fully describe on paper and often change as new ideas develop, prototyping seemed to be a good way to help bring out the necessary level of understanding in this project. Prototyping is basically a process of identifying user needs, developing a working model, demonstrating the model, and continuing the process until the needs are satisfactorily understood by the users and the developers. [8, p. 16].

There are three different ways to approach prototyping according to Andriole. He describes them as evolutionary, throwaway, and hybrid. Evolutionary prototyping means the software developers use the prototype as the starting place for creating the actual software product. Throwaway prototyping means not using any part of the prototype for the actual software. Hybrid prototyping covers the middle ground. [8, p. 15]

The term *storyboard* originally comes from the movie industry where storyboards have been used to determine the requirements for a movie before it is actually filmed [8, p. 39]. In relation to computer software requirements, storyboard prototyping has come to describe a method for interactively viewing a model of an application before it is actually coded [8, p. 34]. In Andriole's words the "essence of all storyboards is the master menu structure," and that there are "two distinct aspects of storyboarding," creation and playback. [8, p. 43] For this project, storyboard creation was necessary—but not playback—since the users wanted to interact with the prototype and not just watch pre-recorded sessions.

3. Prototyping the User Interface: The Process

This project was created using the evolutionary storyboard prototyping method to develop the user interface as well as the underlying functional requirements. This section discusses the tools and the process used for prototyping. Following this discussion is a description of the products of the process.

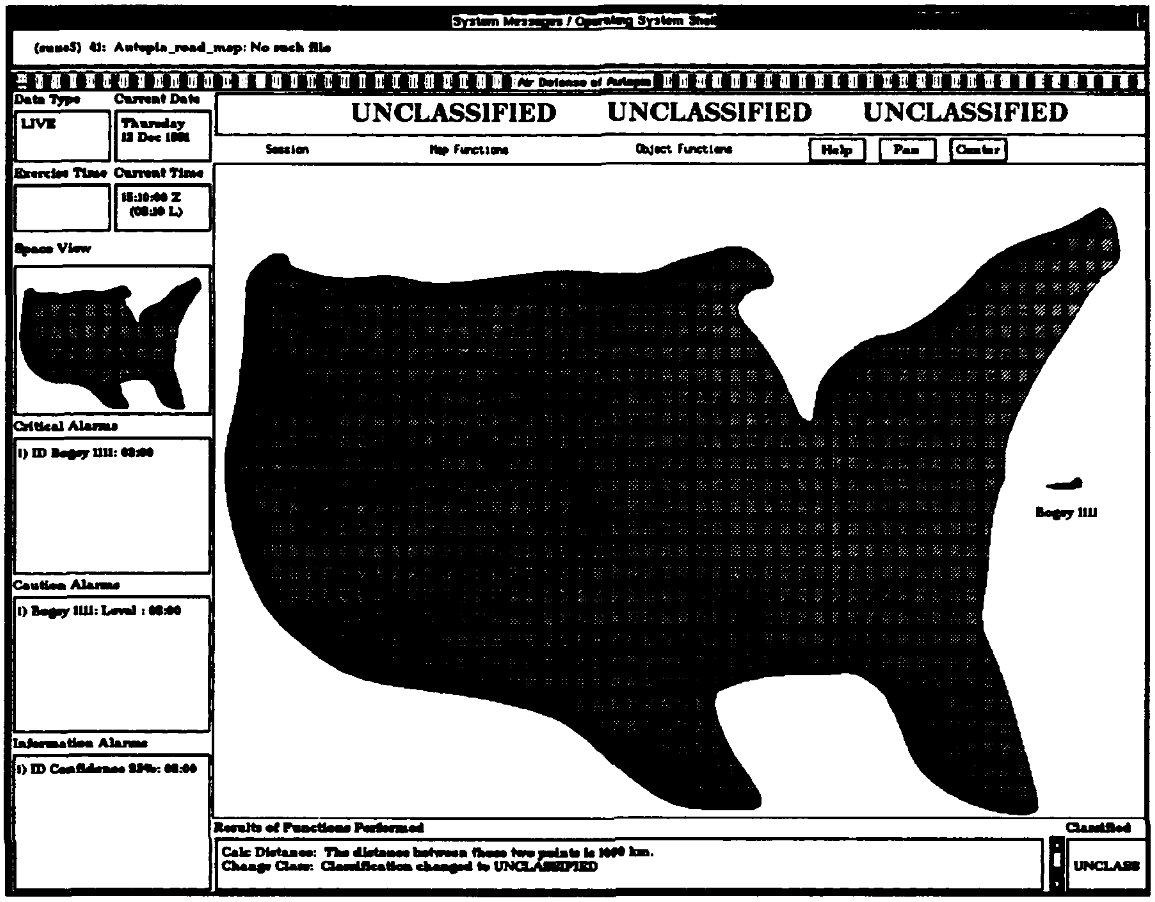
The TAE Plus tool was used because it had impressive prototyping capabilities. TAE Plus provided the means to create, store, rehearse, and edit the user interface. It also generated executable Ada source code for faster execution during critique sessions and for later development of the full application.

In order to make the discussion of TAE Plus more understandable, a short description of TAE Plus terminology is provided. A TAE Plus user interface (storyboard) is made of panels which can contain zero or more items. A panel is basically a configurable, Motif-managed X window. Panels can be connected to each other so that in the rehearse mode, an item selected on one panel can cause another panel to appear. An item is a Motif widget such as a button, icon, or label. Other item explanations will be provide as necessary. The important thing to realize is the storyboards are sets of TAE Plus panels and items which can be interactively rehearsed. Unfortunately, this paper can only provide captured screen images.

a. Creating Initial Storyboards

The first step of this project's prototyping process involved building some initial storyboards based on a reasonable example of a C^2 map-base user interface. The initial and subsequent storyboards were loosely based on the model of the Granite Sentry atmospheric defense user interface. Since the Granite Sentry interface is relatively new, is part of an operational system, and has most of the common functions, it was considered to be a good starting place for a generic map user interface. One of the initial main storyboards is shown in Figure 14. Once it was created, the students and instructor of a human-computer interaction course critiqued it and many of the recommended changes were made. This process was initially done twice. It should be mentioned that Figure 14 is a storyboard from after the first critique but before the second one. A second iteration storyboard is shown since the first one had a "SECRET" label which might have caused unnecessary notice of this thesis.

The main difference between the first iteration storyboard and the one in Figure 14 is the number of separate windows. The first iteration showed basically all of the square areas of the display as separate Motif windows, each of which had a header that wasted space. In Figure 14 only two Motif windows are shown as identified by the bold window headers. Another major change from the first iteration was to divide the alert area into three sections to categorize the importance of threats. It is interesting to note that the Granite Sentry system



(sune5) 41: Autopia_road_map: No such file

Data Type: LIVE

Current Date: Thursday 12 Dec 1991

Exercise Time:

Current Time: 15:10:00 Z (08:10 L)

Critical Alarms

1) ID Bogey 1111: 08:00

Caution Alarms

1) Bogey 1111: Level : 08:00

Information Alarms

1) ID Confidence 85%: 08:00

Results of Functions Performed

Calc Distance: The distance between these two points is 1000 km.
Change Class: Classification changed to UNCLASSIFIED

Classified: UNCLASS

Figure 14. The Initial Main Display Storyboard (second iteration)

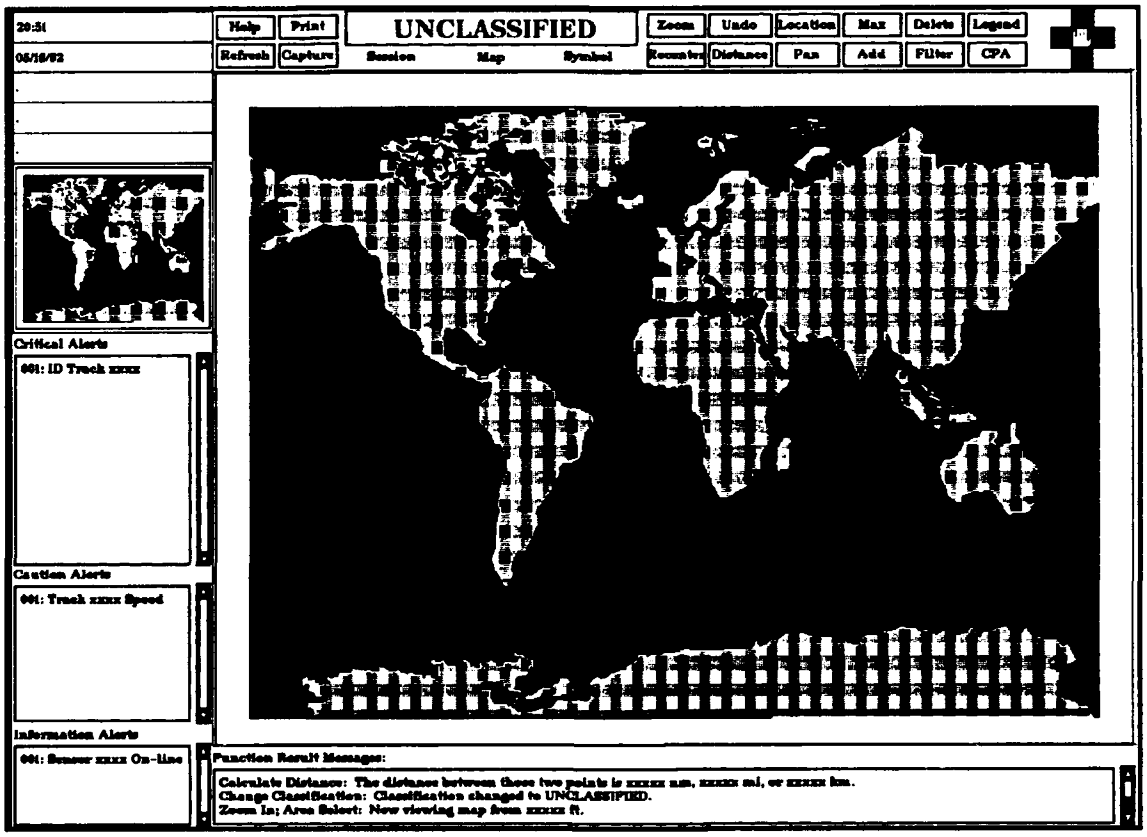
has just over 30 fixed alerts predefined. However, for this more generic user interface, any number of alerts can be displayed on a list which the user can scroll through.

b. Reviewing the Storyboards

The second step of the process was reviewing the storyboards. As just mentioned, the first two review iterations involved informal critiques from fellow students and an instructor. During the second review period the thesis advisors also reviewed and interacted with the storyboards. The advisors also participated in two additional reviews. Changes were made based on their comments and other research findings that came up during the course of analysis.

c. *Creating Multiple Storyboard Versions*

The last step in the prototyping process involved iteration—creating improved versions of the storyboards until an acceptable set was developed. Since there are too many different versions of the storyboards to include in this paper, Figure 15 shows the main storyboard that was accepted as the final main display for the user interface. However, there



001: ID Track xxxx

001: Track xxxx Speed

001: Sensor xxxx On-line

Calculate Distance: The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

Figure 15. The Main Display Storyboard (final iteration)

are other panels not shown which were used for data entry and information display. As Figure 15 shows, the major changes reflected in the final iteration deal mostly with the specific placement of items. The example world map is another noticeable change. Otherwise the basic kinds of information shown by the interface changed little.

4. Prototyping the User Interface: The Products

During the prototyping process, several different products were created to document the requirements. The products included a feature chart, an outline of the main storyboard, and a full set of TAE Plus storyboard panels.

a. The Feature Chart

In general a feature chart is for showing the basic requirements or features for a software application. It also serves as a blueprint for specifying the relationships between the different parts of the software. [49]

The feature chart shown in Figure 16 depicts the user interface requirements in a menu structure, which happens to fit well with Andriole's mention of using a master menu structure for storyboarding. The boxes represent user-interaction items such as buttons, menus, and dialog boxes, and the ovals represent functions that the user can perform. The directed arrows show which functions will be accessed from which items. From the main menu, the session, map, and symbols functions can be selected. The help functions shown under each menu cannot actually be selected from the menus. Rather, they depict the fact that the menu functions have specific help available by using an external help button. The items and functions not connected to the menus are constantly available to the user and therefore do not need to be selected via a menu.

The session menu includes functions for display management and output. The map menu includes the common functions listed in the requirements as well as those helpful features, such as undo. The symbol menu contains the common symbol functions along with other related functions.

b. The User Interface Layout

An outline of the user interface was also created during the prototyping process to document the primary features of the main panel. Figure 17 shows the main panel layout.

The layout shows the initial user panel. From this panel the user can select functions related to the session, map, and symbols. This kind of user interface is generally referred to as event driven. Each action the user performs generates an event to which the interface needs to respond. The response may include displaying another part of the interface,

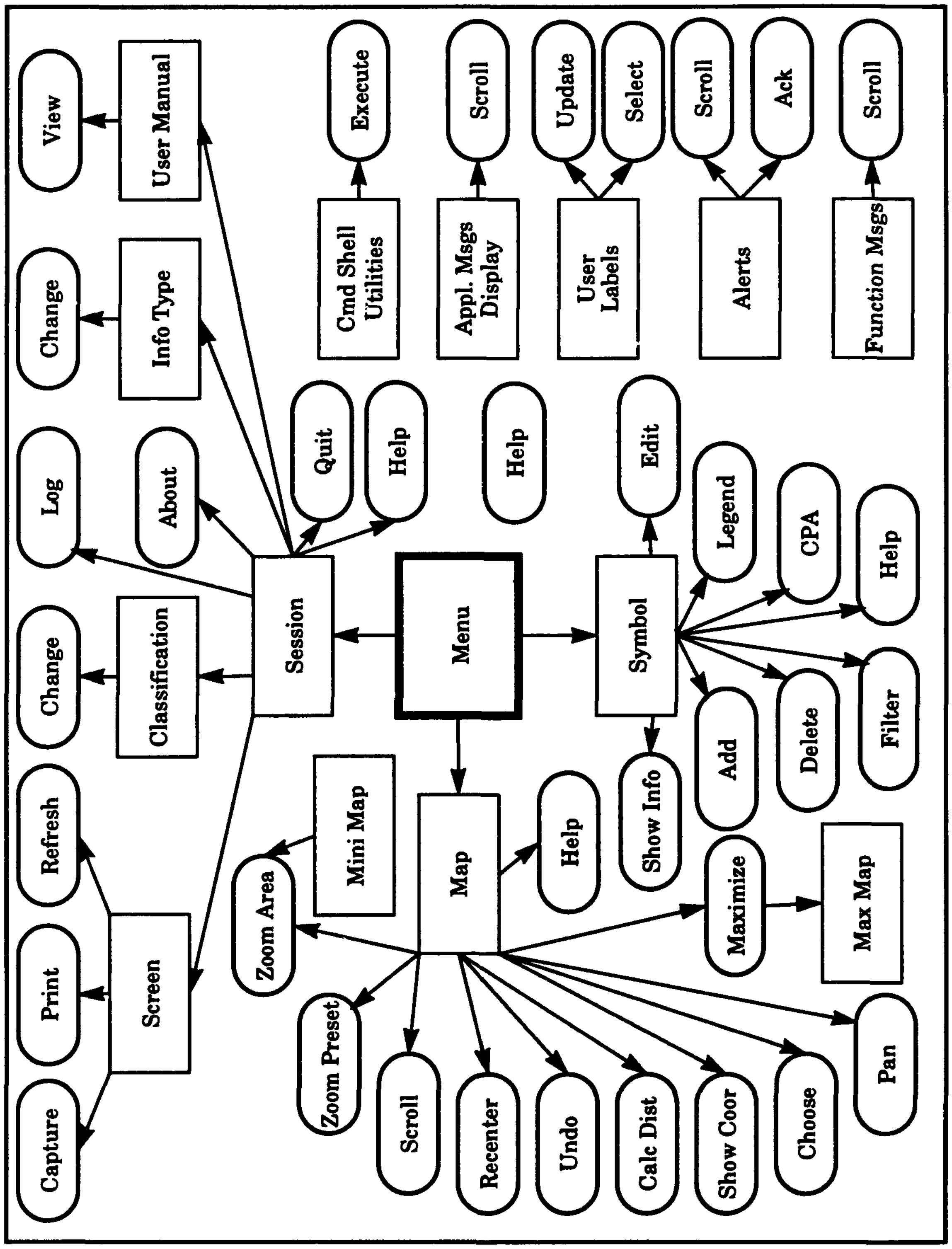
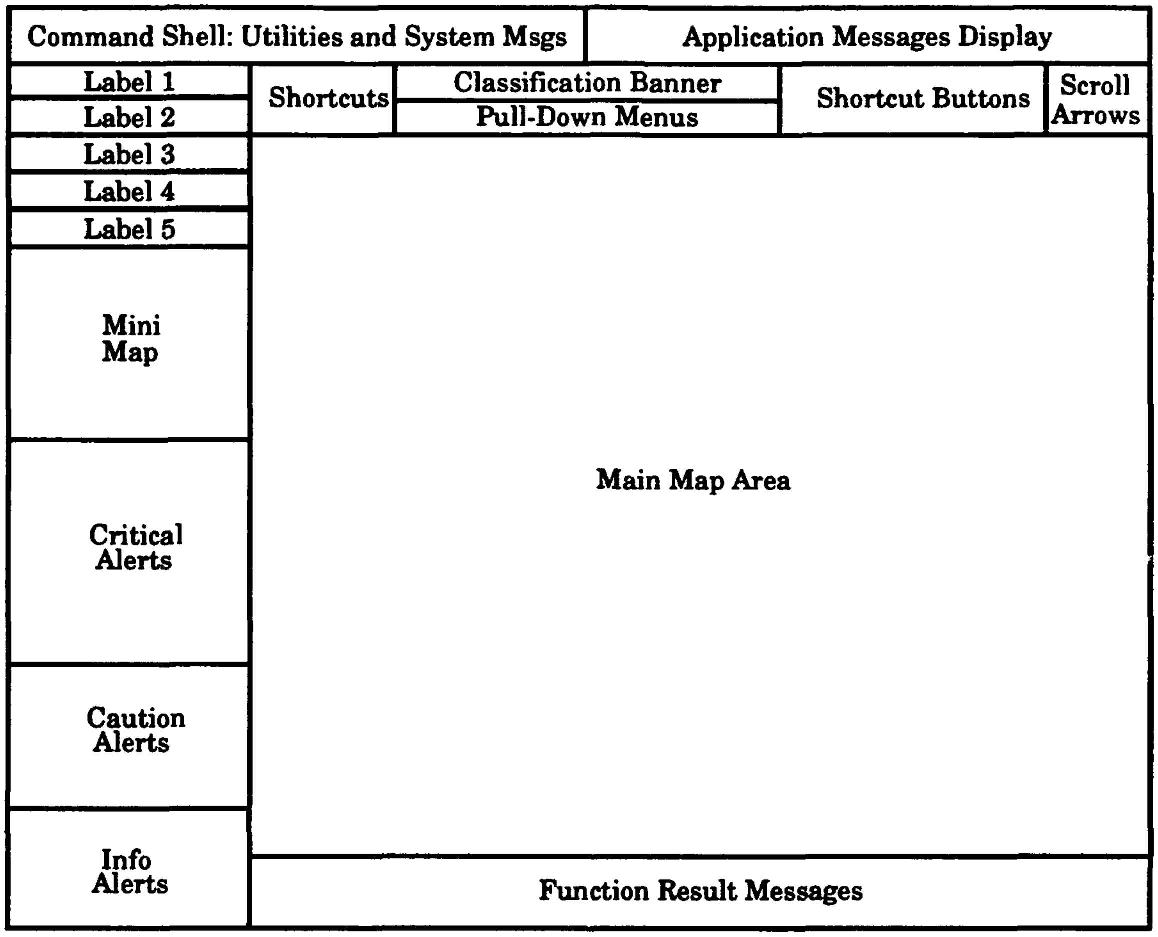


Figure 16. Feature Chart



1. The Command Shell: Utilities window is for invoking operating system programs such as mail, a calculator, etc. Operating system messages can also appear in this window.
2. The Application Messages Output window displays messages from the user interface or application program.
3. Labels 1-5 are client application definable to display whatever information is needed (e.g. time, date, etc.). The user can also select the labels if the client application requires an acknowledgement of the information being displayed.
4. The Classification Banner displays the user or application-selected classification of the information on the display. Currently two levels are provided, unclassified and secret.
5. The Pull-down menus provide access to all of the session, map, and symbol functions.
6. The Shortcut buttons to the left of the banner and menus are for buttons related to the session menu.
7. The Shortcut buttons on the right side are related to the most common map and symbol functions.
8. The Scroll Arrows are for moving only the main map in one of four directions.
9. The Critical, Caution, and Information Alert areas are for appropriate category of messages sent to the interface by the application program.
10. The Function Result Message area displays informational messages mainly generated by the functions invoked by the user.

Figure 17. User Interface Layout

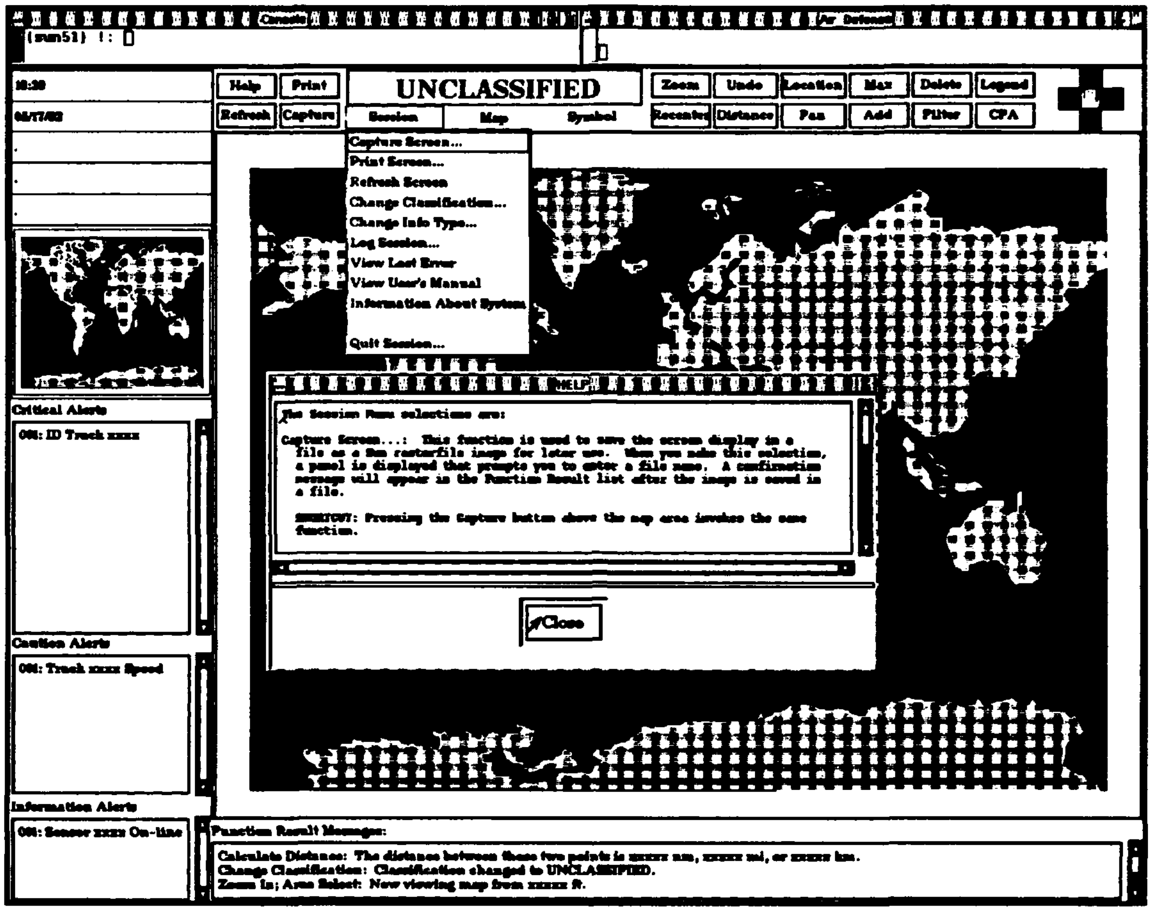
invoking one of the common map or symbol functions, or passing information to the application program.

Starting at the top of Figure 17, the layout shows a window for user access to the operating system and a window for displaying internal application status messages. Below these windows, the five small areas on the left side provide labels in which the client application can display any numeric or textual information that will fit. Also below the top windows are the classification banner and shortcut buttons. The buttons provide shortcuts to the most commonly performed functions listed in the three pull-down menus. The main map area is for displaying the primary view of the map. The scroll arrows are for moving the main map in the direction the user chooses. The mini map area is for showing, if possible, the whole map area. This allows the user to change the main map view (e.g. zoom) while the mini map stays constant to provide a constant, overall perspective of the map area. The three alert areas provide a place for important local and remote messages to be displayed. The function result message area is for displaying the results of certain actions the user takes. The text below the figure describes the layout featured in more detail.

c. *The Final Storyboards*

This section describes the accepted storyboards for the session, the common map functions, and the common symbol functions as shown in the feature chart and listed in Chapter II. As with Figure 17, each storyboard has a brief general textual description as well as the detailed figure description. When a storyboard did not prove useful to show a function, a detailed textual description is given instead.

Before showing the function-specific storyboards, it is helpful to look at the general features of the two main storyboards—the main map panel and the max map panel. Figure 18 shows the first main storyboard which has the main map panel window, and the command shell and application messages windows. The main map panel shows the session function pull-down menu and the help panel for the menu. The help panel is typical of the style of help that TAE Plus provides as part of its development environment. The help text is contained in external files which the application developer can edit to provide specific information. The standard convention for selecting the help function and other functions in this TAE Plus developed user interface is to click the left mouse button once (single-clicking)



The Session Menu selections are:

Capture Screen...: This function is used to save the screen display in a file as a Sun rasterfile image for later use. When you make this selection, a panel is displayed that prompts you to enter a file name. A confirmation message will appear in the Function Result list after the image is saved in a file.

SHORTCUT: Pressing the Capture button above the map area invokes the same function.

Calculate Distance: The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The user will initially see the Main Map panel window and the smaller top windows. The Main Map panel does not have a Motif header which prevents the user from moving or resizing the panel.
2. The Command Shell: Utilities window (with the Console title) allows the user to invoke any external utilities that might be needed. The Motif window functions allow the window to be resized or scrolled if needed. Operating system messages can also be seen in this window.
3. The Application Messages Display window (with the Air Defense title) lets the user see the messages regarding the state of client application and interface. This window can be resized and scrolled like the shell window.
4. From the main panel the user can select functions from one of the three menus or from the shortcut buttons. The session menu is shown as an example.
5. In all of the menus, the three dots following some of the function names indicates there are more panels that the user will see when the function is invoked.
6. All of the help panels can be displayed at the same time that the items they describe are activated. The header on this panel indicates it may be moved to different position or minimized into an icon. However, it cannot be resized.
7. The first two client application-definable labels show the time and date as examples of what kind of information could be displayed there.
8. The user can acknowledge the alert messages and can scroll through the Function Result Messages.

Figure 18. The Main Map with Session Menu and Help

when the pointer is over the intended area. For the application specific functions, there are exceptions, and they will be discussed.

All three of the pull-down menus: session, map, and symbol, list the most frequently used functions towards the top and the least often towards the bottom. The function use frequency was just an estimate. No tests were done to determine an optimum order.

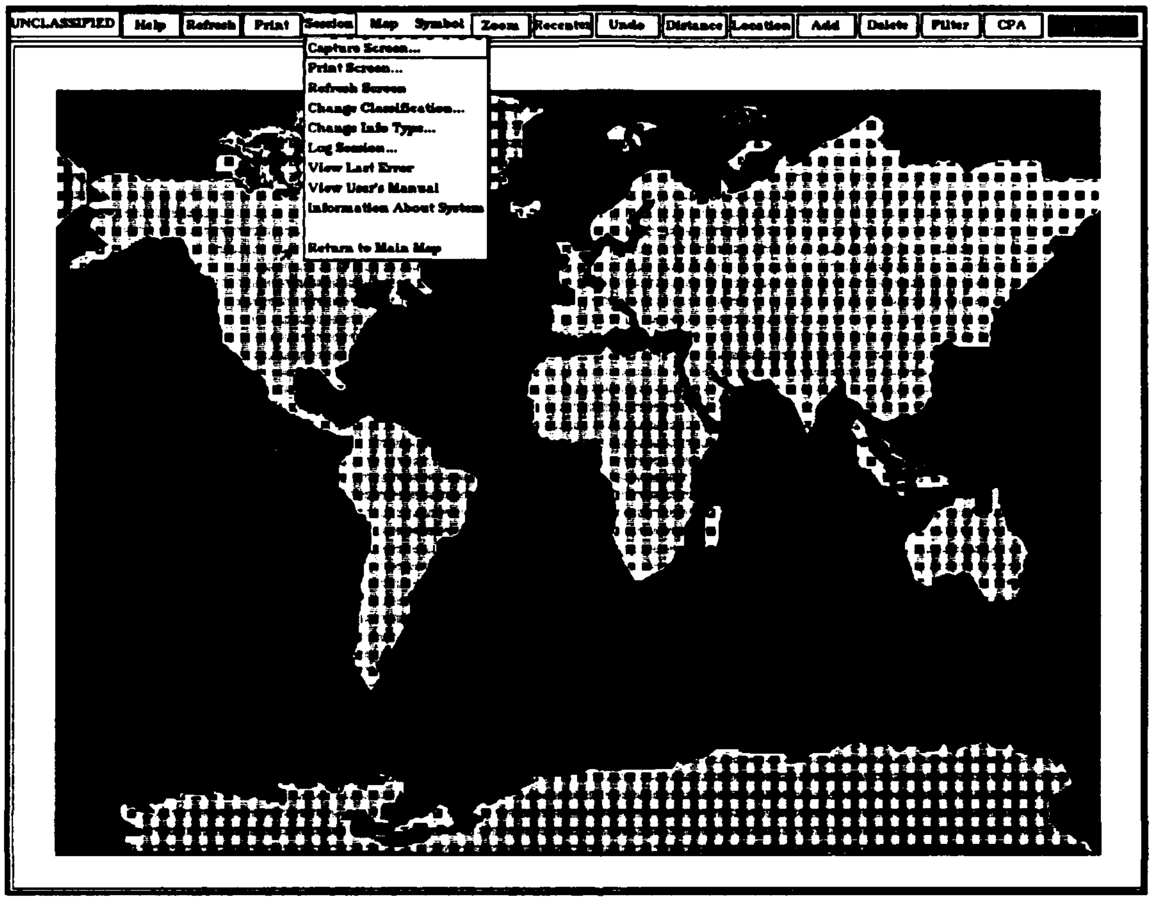
The second main storyboard in Figure 19 shows the maximum-sized map panel. This panel is made visible by selecting the Max button or the Maximize Map Size option in the Map menu on the first main panel.The user can invoke the same functions from the max map panel as from the main map panel except for quitting the session. Of course, the max map panel does not have as many shortcut buttons as the main map panel since there was not enough space. The session pull-down menu shown in Figure 19 has a Return to Main Map selection instead of Quit Session as shown in Figure 18. Otherwise the pull-down menus are identical for both panels.

(1) The Session Functions

The user interface session functions include a combination of the related C^2 and screen and window functions listed in Chapter II.

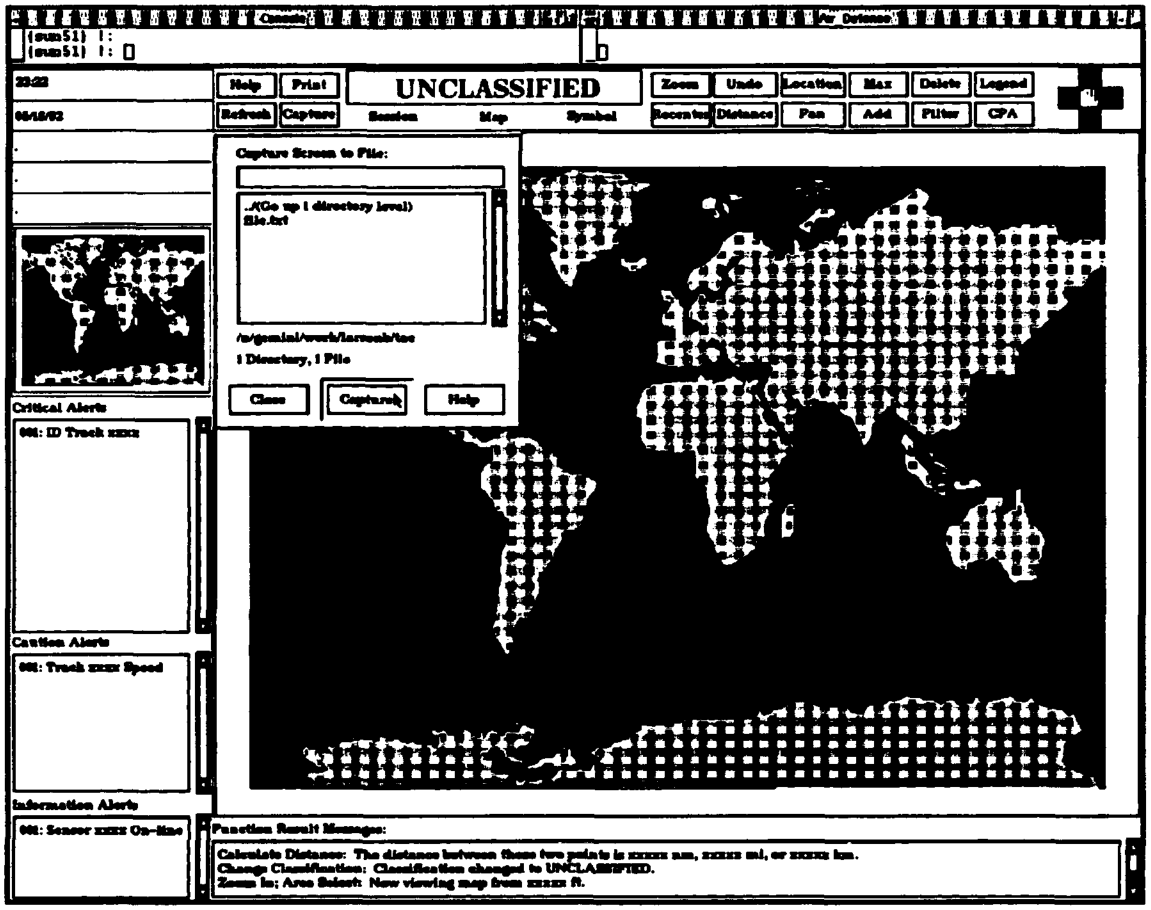
The first function listed in the session menu is Capture Screen. This function captures an image of the whole screen and saves it to a file. This image can be used for later review of some important event or other purpose. Since images (especially color) take up a lot of disk storage space, this function should be used carefully. The Figure 20 storyboard shows the file-prompt panel that the user sees after invoking the Capture Screen function.

Although it is not the next function in the session menu, the Log session function displays essentially the same file-prompt panel as the Capture Screen function. Therefore, Figure 20 also shows the Log Session Panel except that the title is different. The purpose of the Log Session function is to store application-specified historical or exercise information. The information to be saved is specified by the client application. Messages and periodic screen displays are the kinds of information intended to be stored.



1. The user will see this panel after selecting the Maximize Map Size function in the Map menu or after pressing the Max button on the main map panel.
2. The user can perform the same functions on the panel as on the main map panel except to quit the session. The user must first select the Return to Main Map as shown in the session menu.
3. Although it cannot be shown with the example map, the map on this panel is intended show the same map location and scale as the main map panel, but with more area visible around the edges.
4. So that the user does not miss any important information normally shown on the main map panel, a warning message will appear on this panel as determined by the client application.

Figure 19. The Max Map with Session Menu



Calculate Distance: The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The Capture Screen and Log Session data entry panels are identical except for the panel title. However, they cannot be shown at the same time since they overlap. Both functions can be invoked from the session menu. The Capture Screen function also has a shortcut button.
2. The user can enter a file name in the keyin item area below the title or select a file name to overwrite by clicking the left mouse button twice (double-clicking) in the file list area.
3. Double-clicking in the keyin area that contains text will cause the text to be highlighted. If the user then presses any keyboard key, the text will be deleted. This speeds up re-entry of file names. This is a feature commonly found in various commercial software packages such as the FrameMaker desktop publishing system [24]. To undo the highlighting before the text is deleted, the user only needs to single-click the left mouse button. All of the keyin areas in all of the panels work this way.
4. The user can traverse the file directories by entering the full path name in the keyin area or by double-clicking on any displayed directories in the file list area.
5. Pressing the Capture button causes the data entry panel to disappear and then the screen to be captured. Also, a confirmation message will appear in the Function Result list.
6. Pressing the Close button causes the data entry panel to disappear and cancels the function.
7. Pressing the Help button, causes the pointer to look like a question mark. The user can then select any part of the data entry panel to get help information about that area. This help feature works the same way throughout all of the panels.
8. If a function-related error occurs, a panel with an error message will appear.

Figure 20. Capture Screen and Log Session

The next session menu function is Print Screen. This function captures the screen image for direct printing or for storage as a file in the industry-standard PostScript print format. Figure 21 shows the panel that prompts the user for a printer and/or file name.

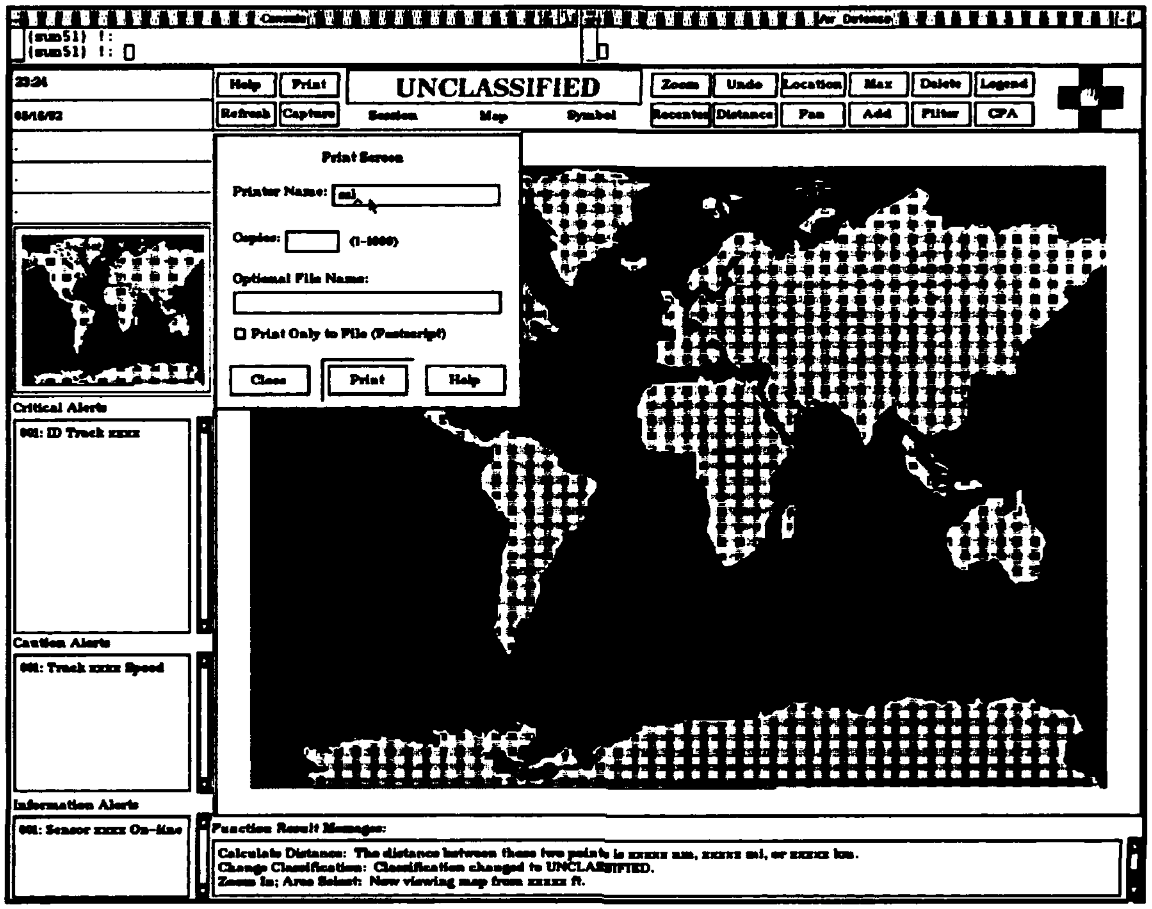
The session function that is next is Refresh Screen. This function does not have a storyboard since it is executed immediately after being selected. The purpose of this function is to redraw the screen if any windows become cluttered with extraneous "junk." Even though X is a pretty stable windowing system, sometimes the X client applications allow certain things to be displayed that should not be. Currently, this is a common part of workstation windowing environments. Refreshing is not often needed, usually.

The function, Change Classification, is the next session menu function. This function allows the user to manually change the classification label to either Unclassified or Secret. If the user selects Unclassified when Secret information is displayed on the screen, then that information is temporarily removed until the classification is changed back to Secret. This function can also be automatically invoked by the client application. In this way other classification levels can be supported. This function is useful for helping to "sanitize" workstations in a secure room for visitors or other reasons. Figure 22 shows the panel the user sees after invoking the function.

The Change Info Type function is the next session menu selection. This function is used to determine what kind of information the user interface should display. The user can then use the same workstation for training with exercise information or use it for real or "live" information. The client application would determine what actual information to display. Figure 23 shows the panel the user sees after invoking this function.

The View Last Error function in the session menu allows the user to display the last function error message that the application reported. This allows the user to review an error after having previously closed the error panel in order to uncover the map to see what the problem might have been. The error panel shown in Figure 24 allows plenty of space for the application developer to provide meaningful messages.

Two more session functions are combined onto one storyboard. They are View User's Manual and Quit Session. The View User's Manual function allows the application



Print Screen

Printer Name: sel

Copies: (1-1000)

Optional File Name:

☐ Print Only to File (Postscript)

001: ID Track xxxx

001: Track xxxx Speed

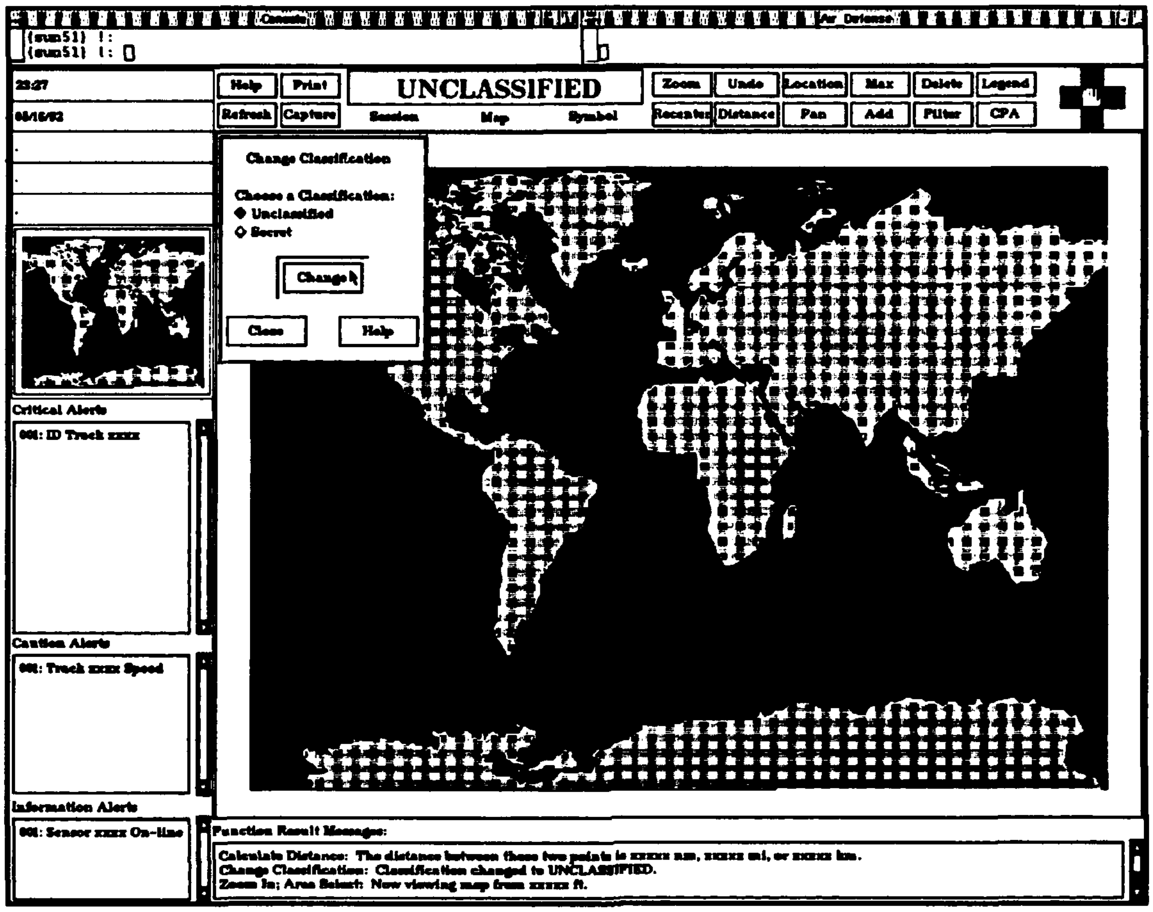
001: Sensor xxxx On-line

Function Result Messages:

Calculate Distance: The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The Print Screen data entry panel prompts the user for a printer name, the number of copies wanted, and a file name. The function can be invoked from the menu or the shortcut button.
2. The screen can be can be sent to only the printer, to both the printer and a file, or just to a file depending on the entries made.
3. Pressing the Print button causes the data entry panel to disappear and then the screen to be printed. Also, a confirmation message will appear in the Function Result list.
4. Pressing the Close button causes the data entry panel to disappear and cancels the function.
5. If a function-related error occurs, a panel with an error message will appear.

Figure 21. Print Screen



(sun51) !:
(sun51) !:

23:27
08/16/92

Change Classification

Choose a Classification:
◆ Unclassified
◇ Secret

Critical Alerts
001: ID Track xxxx

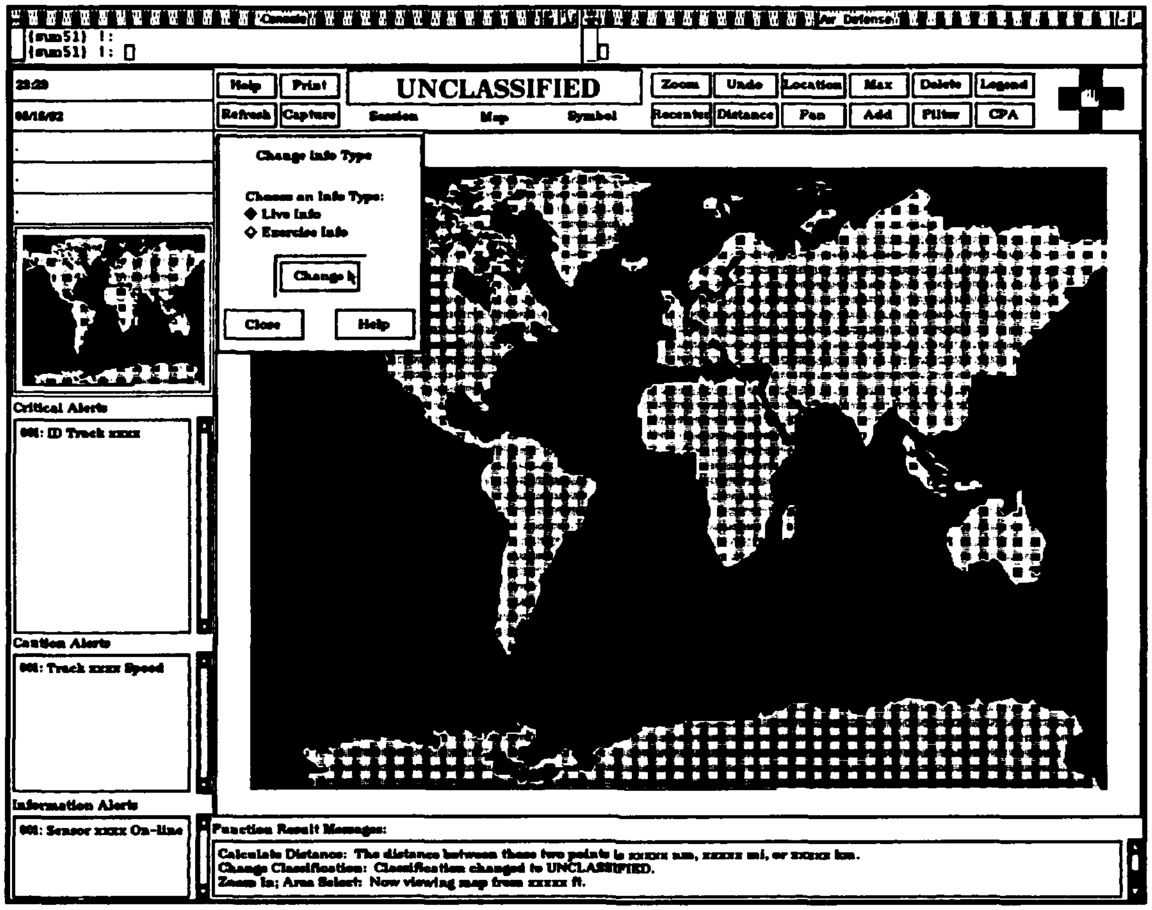
Caution Alerts
001: Track xxxx Speed

Information Alerts
001: Sensor xxxx On-line

Function Result Messages:
Calculate Distance: The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The Change Classification data selection panel allows the user to choose either the Unclassified or the Secret radio button. The user can also display this panel by single-clicking on the classification banner which is above the three pull-down menus.
2. Pressing the Change button causes the data selection panel to disappear and then the classification to be changed. Also, a confirmation message will appear in the Function Result list.
3. Pressing the Close button causes the data selection panel to disappear and leaves the current classification unchanged.

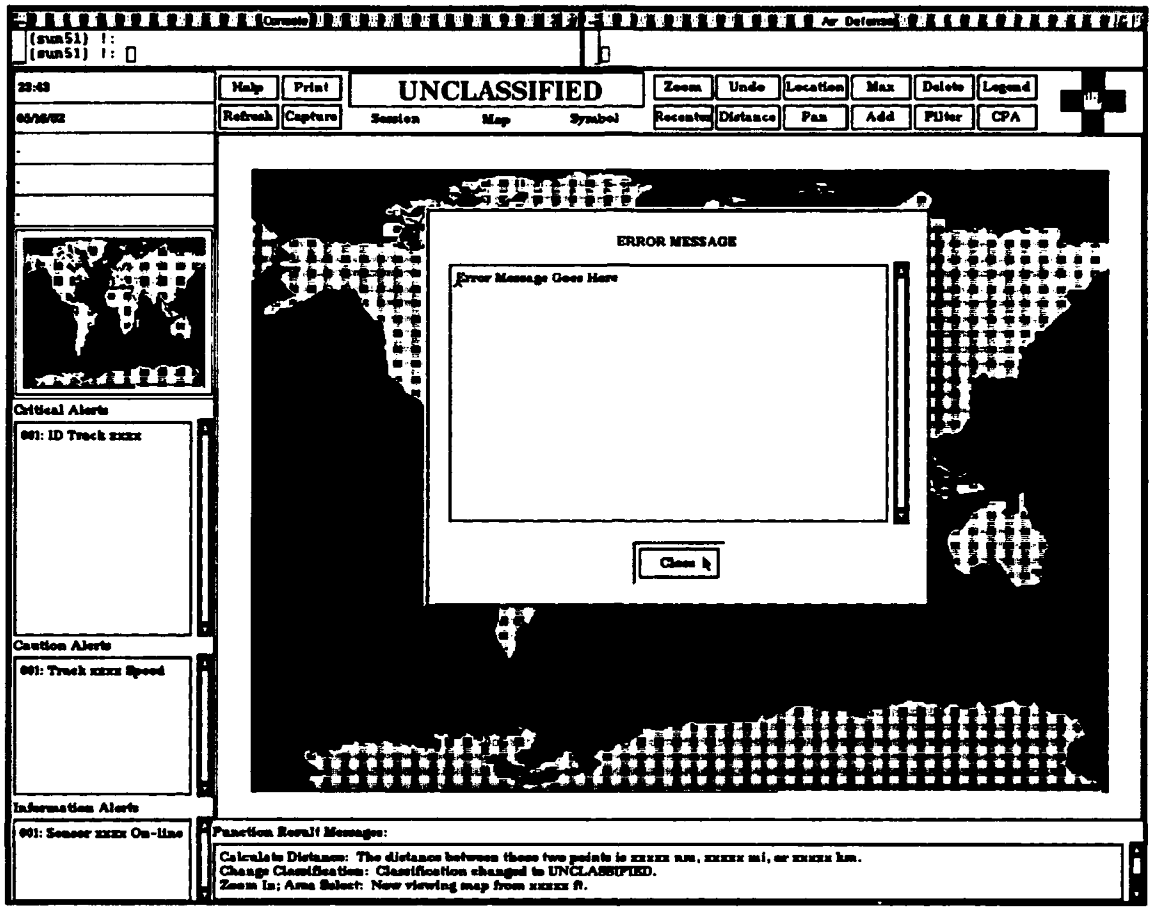
Figure 22. Change Classification



```
Calculate Distance:  The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification:  Classification changed to UNCLASSIFIED.
Zoom In; Area Select:  Now viewing map from xxxxx ft.
```

1. The Change Info Type data selection panel allows the user to choose either the Live or Exercise radio button.
2. Pressing the Change button causes the data selection panel to disappear and then the information type to be changed. Also, a confirmation message will appear in the Function Result list.
3. Pressing the Close button causes the data selection panel to disappear and leaves the current information type unchanged.

Figure 23. Change Information Type



Error Message Goes Here

001: ID Track xxxx

001: Track xxxx Speed

001: Sensor xxxx On-line

Calculate Distance: The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The Error Message panel shows the user the last error message displayed by the application program. This is the same panel that the application uses to display an error for the first time.
2. Pressing the Close button causes the message panel to disappear.

Figure 24. View Last Error

programmer to add a free-format on-line user's manual. The text can be added to a file without any user interface changes, similar to the way text for the help feature is edited. The Quit Session provides the way for the user to exit the user interface and application. Figure 25 shows this storyboard.

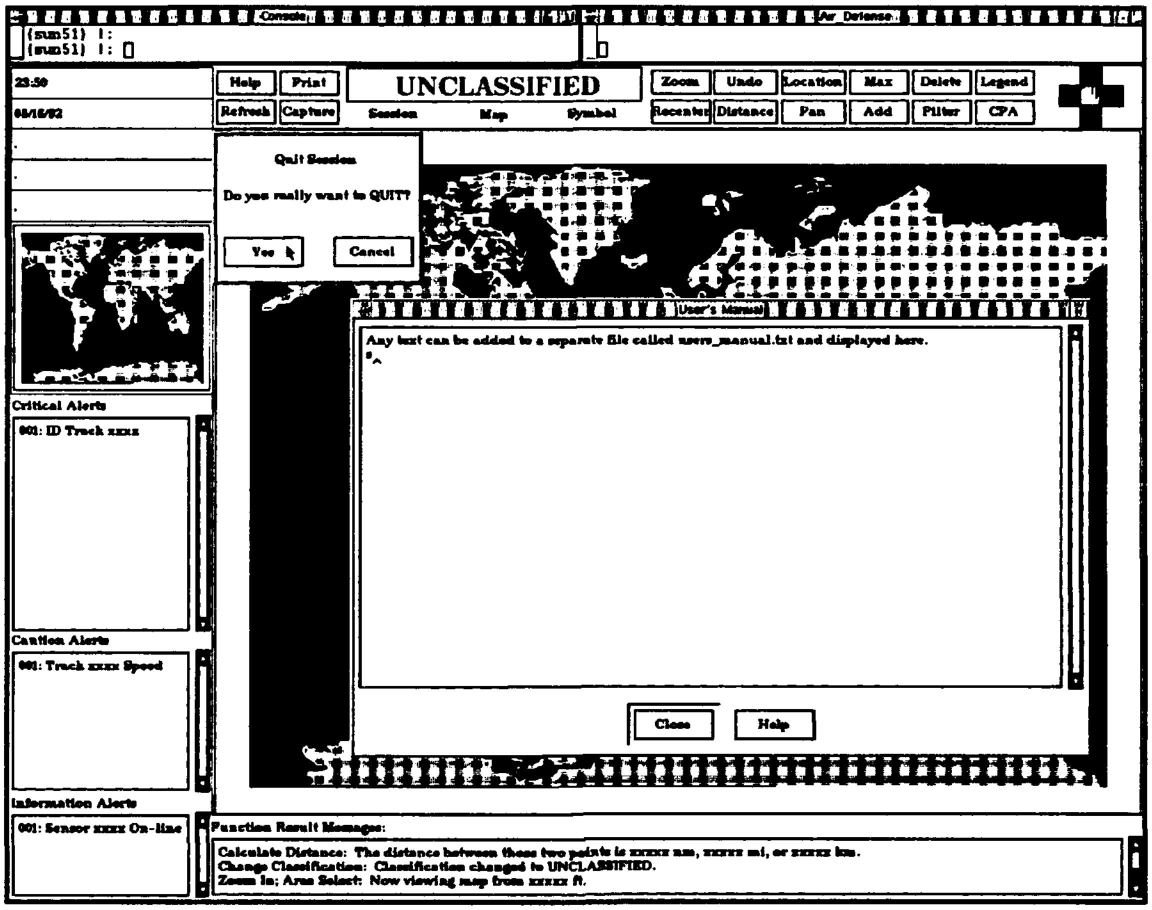
The final session function is Information About System. This function simply displays textual information about what general capabilities the application and user interface provide to the user. This is a common feature found in PC windowing environments. Since this is a very basic function that displays a panel very similar to the User's Manual, no storyboard is provided.

(2) The Map Functions

The map functions in the user interface menu correspond almost directly to the common map functions listed in Chapter II. The map pull-down menu and the help panel for the menu are shown in Figure 26. All of these map functions affect the main and max map in the same way. The mini map only reflects the results of a few functions.

The first function listed in the map menu is Zoom In: Select Area. The purpose of this function is to display a user-selected area of the main map in more detail. The text and help panel shown in Figure 26 describe the detailed procedure for selecting the area to zoom. Invoking this function does not display any panels so there is no separate storyboard. Instead, the pointer changes shape to indicate the zoom function has been selected. Once an area is zoomed, the mini map displays a small rectangle that encloses the area displayed in the main map area. This way the user can keep an overall view of the map. Although it might be beneficial to allow the same Zoom function to be done on the mini map, thus letting different and larger areas of the map be displayed, this added functionality was not considered a requirement for this software.

The next function in the map menu is Zoom to Preset Altitude. The purpose of this function is to allow the user to select a specific map scale from a predetermined list. Figure 27 shows the panel with some example scale choices. This kind of function was used in the Granite Sentry interface primarily because the operators only needed certain map scales



```
(sun51) 1:
(sun51) 1:
```

23:50

08/18/92

Quit Session

Do you really want to QUIT?

Any text can be added to a separate file called users_manual.txt and displayed here.
s^

001: ID Track xxxx

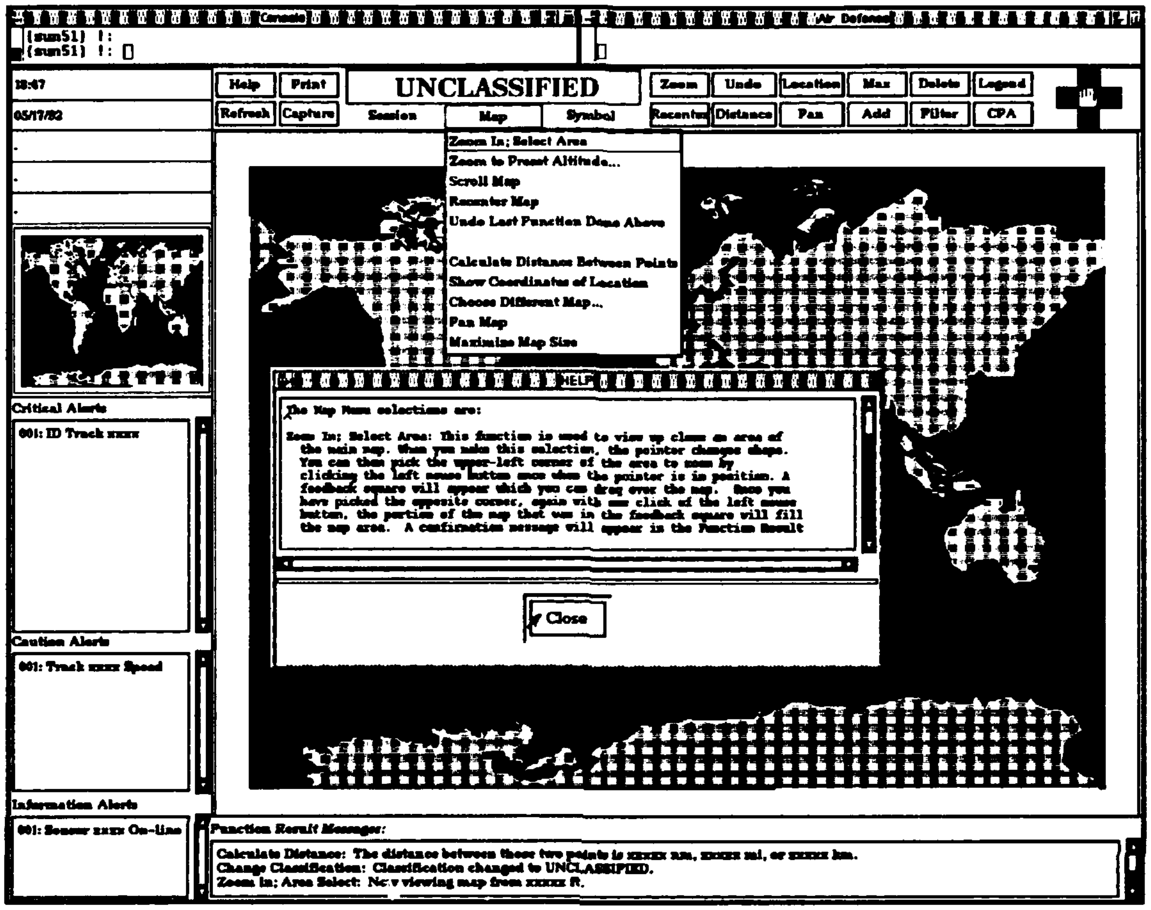
001: Track xxxx Speed

001: Sensor xxxx On-line

Calculate Distance: The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The User's Manual panel shows the user the on-line version of the application's manual. Since this panel has a window header, it can be resized and moved so as not to hide an area the user wants to see at the same time as the manual.
2. Pressing the Close button causes the manual panel to disappear.
3. The Quit Session panels allows the user to quit the application.
4. Pressing the Yes button causes the application to quit.
5. Pressing the Cancel button causes the quit panel to disappear and the application running is left running.
6. A note about TAE Plus: The user manual panel shows the character "s" and a caret symbol on the second line because there are still a few minor glitches with TAE Plus. The character is not in the user manual file so the reason for it showing up is not known. The caret symbol is the result of clicking on the manual panel. Even though the text display area is not suppose to show being selected, it still does. These problems are relatively minor and can easily be overlooked.

Figure 25. View User's Manual and Quit Session



The Map Menu selections are:

Zoom In; Select Area: This function is used to view up close an area of the main map. When you make this selection, the pointer changes shape. You can then pick the upper-left corner of the area to zoom by clicking the left mouse button once when the pointer is in position. A feedback square will appear which you can drag over the map. Once you have picked the opposite corner, again with one click of the left mouse button, the portion of the map that was in the feedback square will fill the map area. A confirmation message will appear in the Function Result

Function Result Messages:

Calculate Distance: The distance between those two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The map menu lists the functions in two groups. The top group of functions can be reversed using the Undo function. Since the bottom group of functions do not change the state of the map, they do not need the Undo function.
2. The displayed menu help panel explains the procedure for selecting an area to Zoom In. It basically involves enclosing an area with a box that is proportional to the map display area.

Figure 26. Map Menu with Zoom Area Help

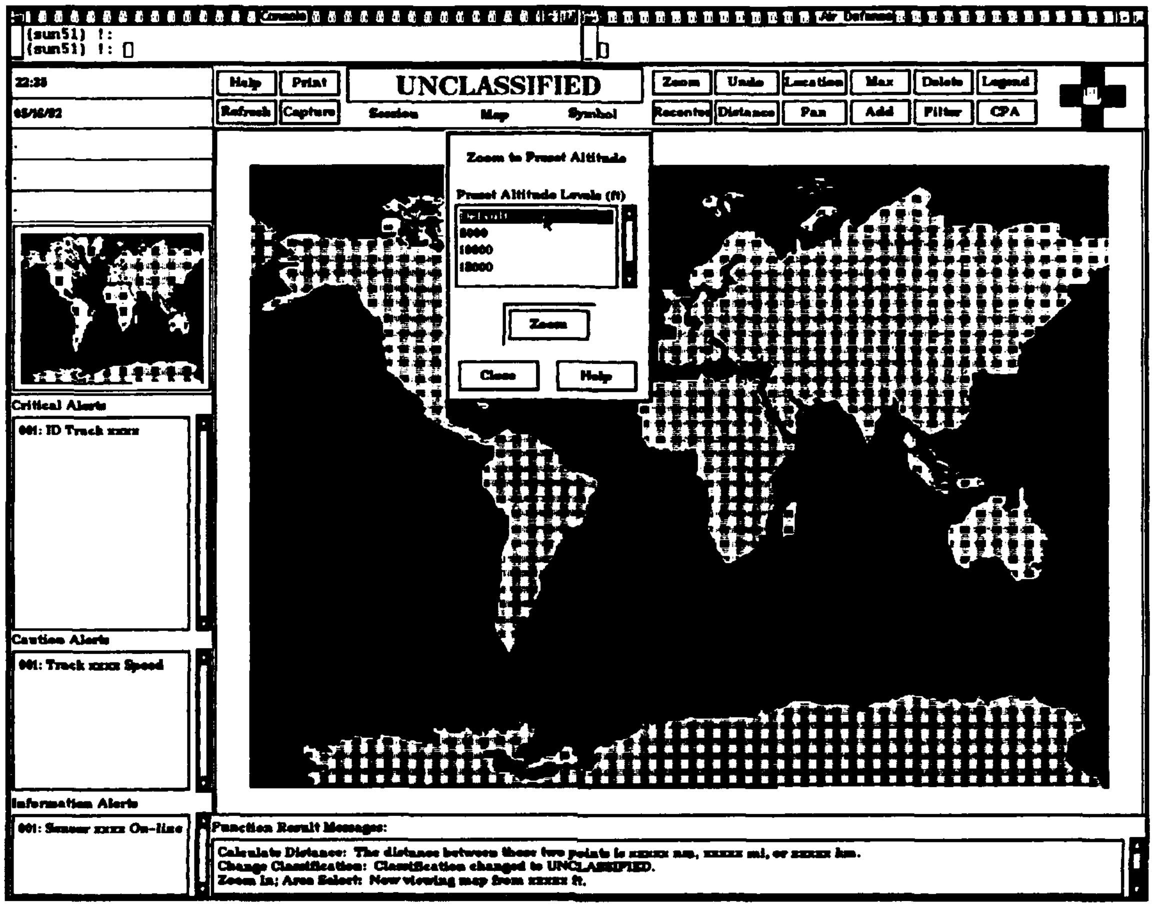
to get the job done. Moving between map views also remains consistent which the area zoom function cannot guarantee.

The Scroll Map function is the next function listed in the menu. This function is to allow the user to change the view area of the map without changing the scale. The menu scroll function causes the pointer to change shape and allows the user to "hook" the map and move it in any sideways direction (vertically, horizontally, diagonally, etc.). Hooking the map involves holding down the left mouse button while the pointer is on the map. The user can then drag the map to a new position. If the pointer reaches one of the sides and more scrolling is wanted using this function, the user must release the mouse button, select the scroll function again, and repeat the process. The same function can be invoked by pressing the "hand" button located in the center of the scroll arrows. A hand icon is used since this kind of scrolling function is like placing a hand on a piece of paper and moving it in any direction. For a different kind of scrolling control, the user can press the scroll arrows which are always active, but which only allow vertical and horizontal movement. By continuously pressing the arrows, the scrolling will continue. The scrolling hand and arrows can be seen in the upper-right corner of any storyboard figure. Also, any zoomed area rectangle displayed on the mini map will move along with the main map.

The next map function is Recenter. This function is used to recenter and reset the main and max map to the default map position and scale. Since this function is executed immediately, no storyboard is provided. This function can also be invoked with the shortcut button.

The Undo Last Function Done Above function shown next in the map menu is used to undo the operation of only the last function that was executed. As shown in the map menu in Figure 26, the only functions that can be undone are listed above the undo selection. For example, if the last map function the user invoked was Recenter Map, the Undo function will only reposition and rescale the map to its previous position—it cannot undo yet another function. The same Undo function can also be invoked by pressing the shortcut button.

The next function, Calculate Distance Between Points, is used to determine the distance between two points on the main or max map. The points may be fixed or moving.



```
(sun51) !:
(sun51) !:
```

22:35

05/16/92

001: ID Track xxxxx

001: Track xxxxx Speed

001: Sensor xxxxx On-line

Calculate Distance: The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The Zoom to Preset Altitude panel shows a list of example altitudes which can be selected. The client application program can display different and more values than shown. A scroll-bar is provided to view more values.
2. The user can select an altitude in one of two ways. The first way is to single-click on one of the values and then press the Zoom button. The panel will then disappear and the function will be executed. The second way is to double-click the value. This automatically removes the panel and executes the function.
3. If the user wants to reverse the effect of the function, selecting the Undo menu item or pressing the Undo button will return the map to the previous state.
4. Pressing the Close button causes the panel to disappear and cancels the function.

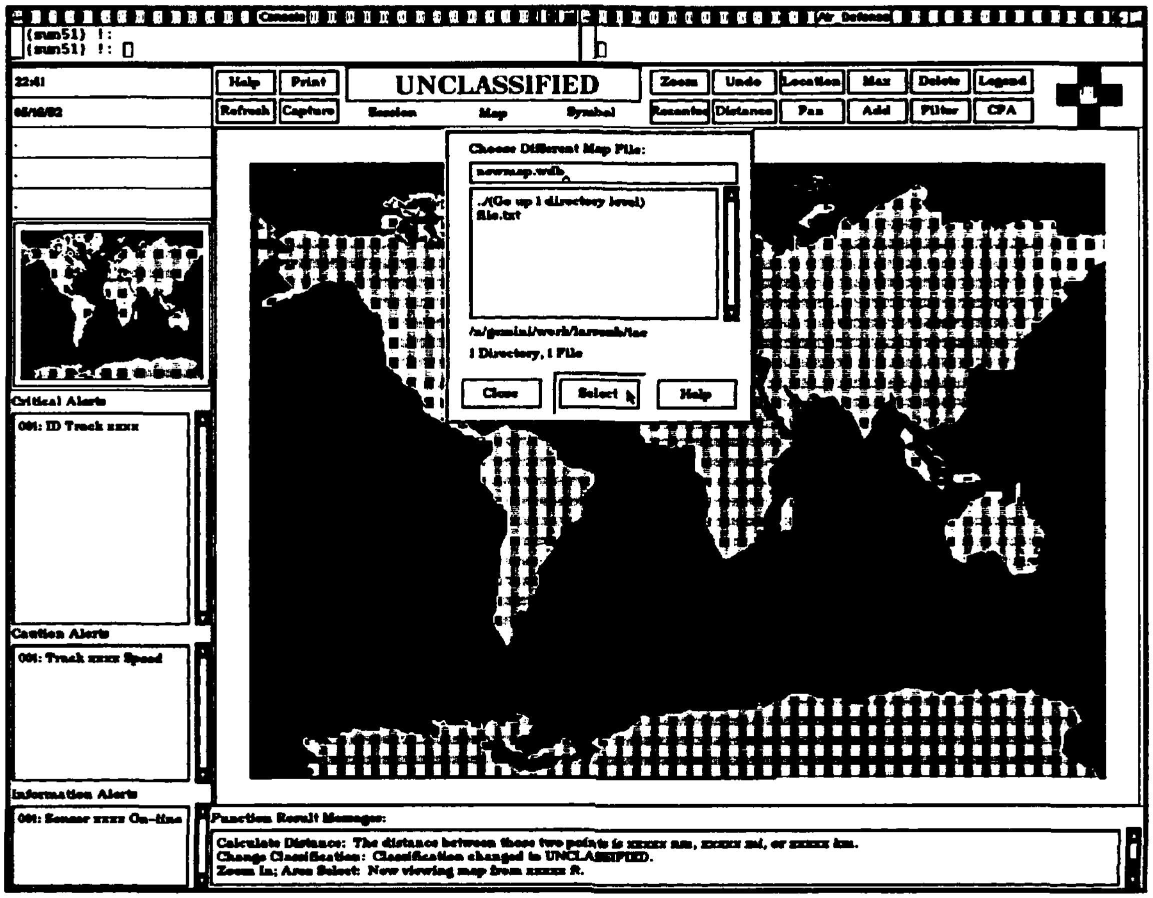
Figure 27. Predetermined Zoom Levels

After the user invokes this function, the pointer's shape is changed which indicates the function is active. The user picks the two points in a way similar to how the corners are picked for the area zoom function. The first point is picked by single-clicking the left mouse button. If a new first point is wanted, a single-click of the right mouse button will cancel the first point. The user can then select a new point without invoking the distance function again. If the user wants to cancel the whole function. a double-click of the right mouse button must be done. Once the first point is selected, the second point is then selected in the same way by single-clicking the left mouse button. After the second point is chosen, the distance is calculated and the result is displayed in the Function Result list. This function can also be invoked by pressing the shortcut button.

The next function is Show Coordinates of Location. The purpose of this one is to display the latitude and longitude of the point selected. After the user invokes the function and the pointer shape is changed, a location can be selected. To show a point's location, the user just needs to single-click the left mouse button. To cancel this function, the user can single-click the right mouse button. The coordinates will be displayed in the Function Result list. This function can also be invoked by pressing the shortcut button.

The Choose Different Map function is the next one listed in the map menu. This function allows the user to select a different map to be displayed. Figure 28 shows the data entry panel that is displayed after the function is invoked. This panel is practically identical to the data entry panels used for the Capture Screen and Log Session functions shown in Figure 20. The user is prompted for a file name of another map. The main, mini, and max map areas will all display the same new map.

The Pan Map function allows the user to select a moving symbol to be the center of the map display area. In this mode, the map automatically scrolls to keep the symbol centered. This is a function commonly required on a shipboard system that needs to track its own-ship [56]. After the user invokes the function and the pointer shape is changed, a symbol can be selected by single-clicking the left mouse button. To cancel this function, the user can single-click the right mouse button. This function can also be invoked by pressing the shortcut button.



Choose Different Map File:

newmap.wdb

./(Go up 1 directory level)
file.txt

/u/gemini/work/larsen/tac

1 Directory, 1 File

001: ID Track xxxx

001: Track xxxx Speed

001: Sensor xxxx On-line

Function Result Messages:

Calculate Distance: The distance between those two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The Choose Different Map File panel prompts the user for a file name of another map.
2. Like the Capture Screen and Log Session panels, the user can enter a file name in the keyin area below the title, as the newmap.wdb example shows. The Select button must then be pressed. A file can also be selected from the file-list display by double-clicking on the file name. In this case the function is executed automatically so the Select button does not have to be pressed.
3. After a new map is loaded, the main and mini map will show the same new map. The max map will also show the same map if selected.
4. Pressing the Close button causes the panel to disappear and cancels the function.

Figure 28. Choose Different Map

The last function in the map menu is Maximize Map Size. The user can display a larger area map with this function. The map area displayed is the same area as shown in the main map area with additional area visible around the edges. Figure 19 shows the Max map. Note that the sample world map does not accurately show display area differences. As previously mentioned, the functions available when this map is displayed are the same as when the main map is displayed, except for Quit Session.

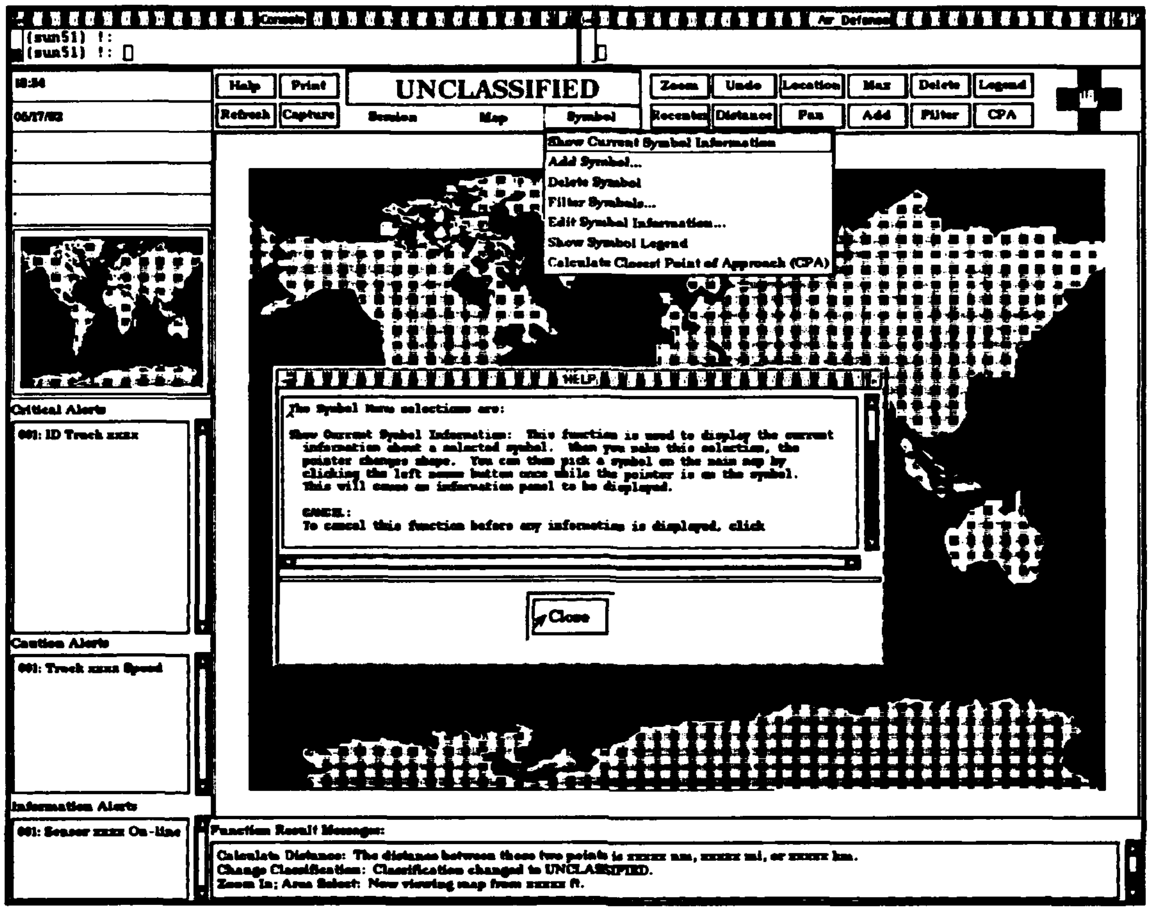
(3) The Symbol Functions'

Unlike the map functions, the symbol function terminology in the user interface is somewhat different than that of the Chapter II function list. The user interface divides the common Show Information function into two functions: Show Current Symbol Information and Show Symbol Legend. Show Current Symbol Information displays the current data for selected symbols while Show Symbol Legend displays static application symbol information. Likewise, the common Edit Symbol function is divided into three functions. Add Symbol, Delete Symbol, and Edit Symbol Information. As mentioned previously and shown in Figure 18, these functions are listed in order of estimated most frequent use. Combining common symbol functions was also done. The common Display Symbol and Remove Symbol functions were consolidated into one Filter Symbols function for the user interface. The two common functions are listed separately in Chapter II because not all of the sources combined them into one function. The following paragraphs describe each of the symbol function storyboards.

The symbol pull-down menu and the help panel for the menu are shown in Figure 29. Some of these functions can also be invoked using the shortcut buttons placed to the right of the menu.

The first function in the list, Show Current Symbol Information, is for displaying the most recent information about a symbol on the map. Figure 30 shows the panel this is displayed when the function and a friendly air track are selected.

The Add Symbol Function allows the user to add symbol manually that, for whatever reason, is not shown on the map from external sensors. Overlays and fixed-position symbols can be added along with tracks. The storyboard only shows the Add Track example

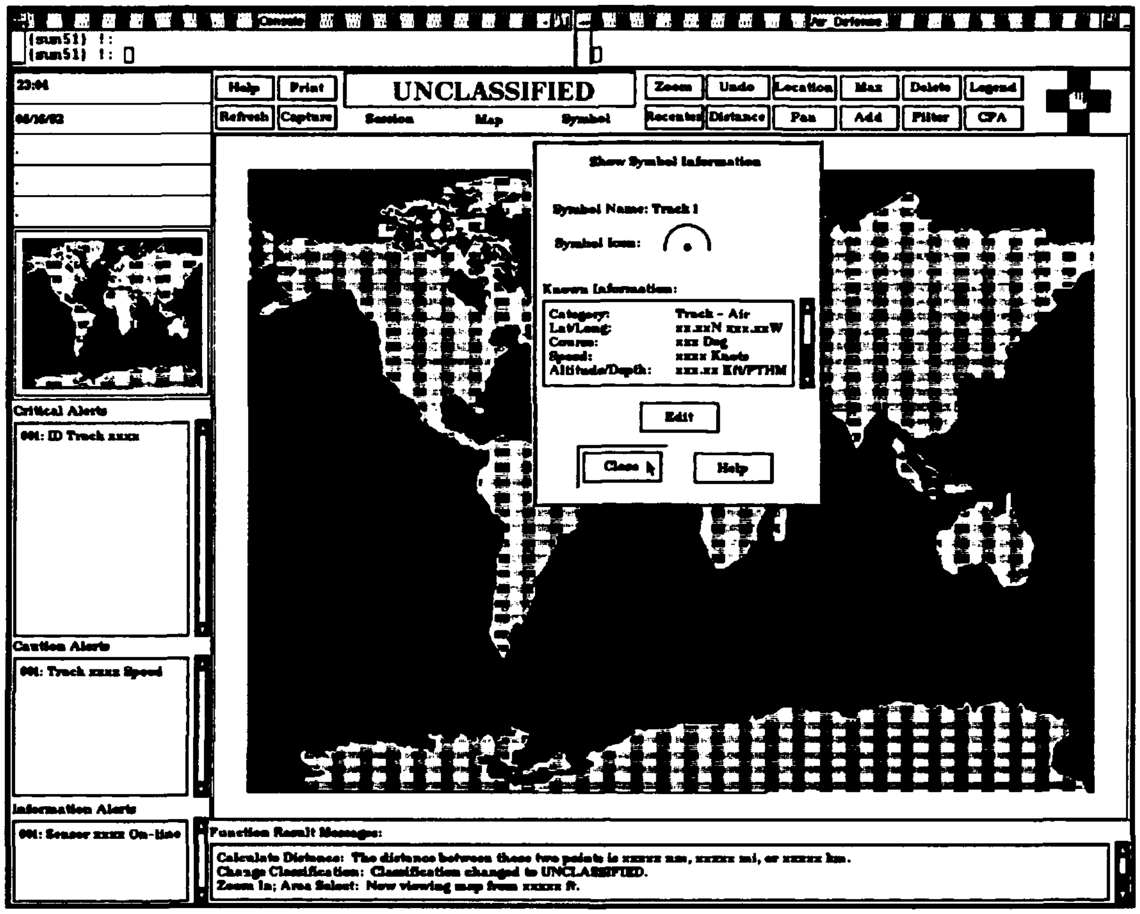


```
The Symbol Menu selections are:

Show Current Symbol Information:  This function is used to display the current
  information about a selected symbol.  When you make this selection, the
  pointer changes shape.  You can then pick a symbol on the main map by
  clicking the left mouse button once while the pointer is on the symbol.
  This will cause an information panel to be displayed.

  CANCEL:
  To cancel this function before any information is displayed, click
```

Calculate Distance: The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The Symbol menu and the menu help panel are shown in this storyboard.
2. The Add Symbol, Delete Symbol, Filter Symbols, Show Symbol Legend, and Calculate Closest Point of Approach also have corresponding shortcut buttons, Add, Delete, Filter, Legend, and CPA.
3. Symbols are displayed on the max map but not on the mini map.

Figure 29. The Symbol Menu with Help



Show Symbol Information

Symbol Name: Track 1

Symbol Icon:

Known Information:

Category:	Track - Air
Lat/Long:	xx.xxN xxx.xxW
Course:	xxx Deg
Speed:	xxxx Knots
Altitude/Depth:	xxx.xx Kft/FTHM

001: ID Track xxxx

001: Track xxxx Speed

001: Sensor xxxx On-line

Calculate Distance: The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The Show Symbol Information panel displays the symbol icon and lists relevant symbol information. The symbol panel also allows the user to select the Edit function for the particular symbol.
2. The user can display this information only by double-clicking the left mouse button when the pointer is on a specific symbol. No function needs to be previously selected.
3. The user can scroll the list of known information to look at additional values.
4. Pressing the Close button causes the panel to disappear.

Figure 30. Show Current Symbol Information

panel in Figure 31. Except to allow the user to input an initial location and heading for a manually entered track, an updating function is not provided. Adding an overlay requires an external graphics editor to draw user-defined symbols.

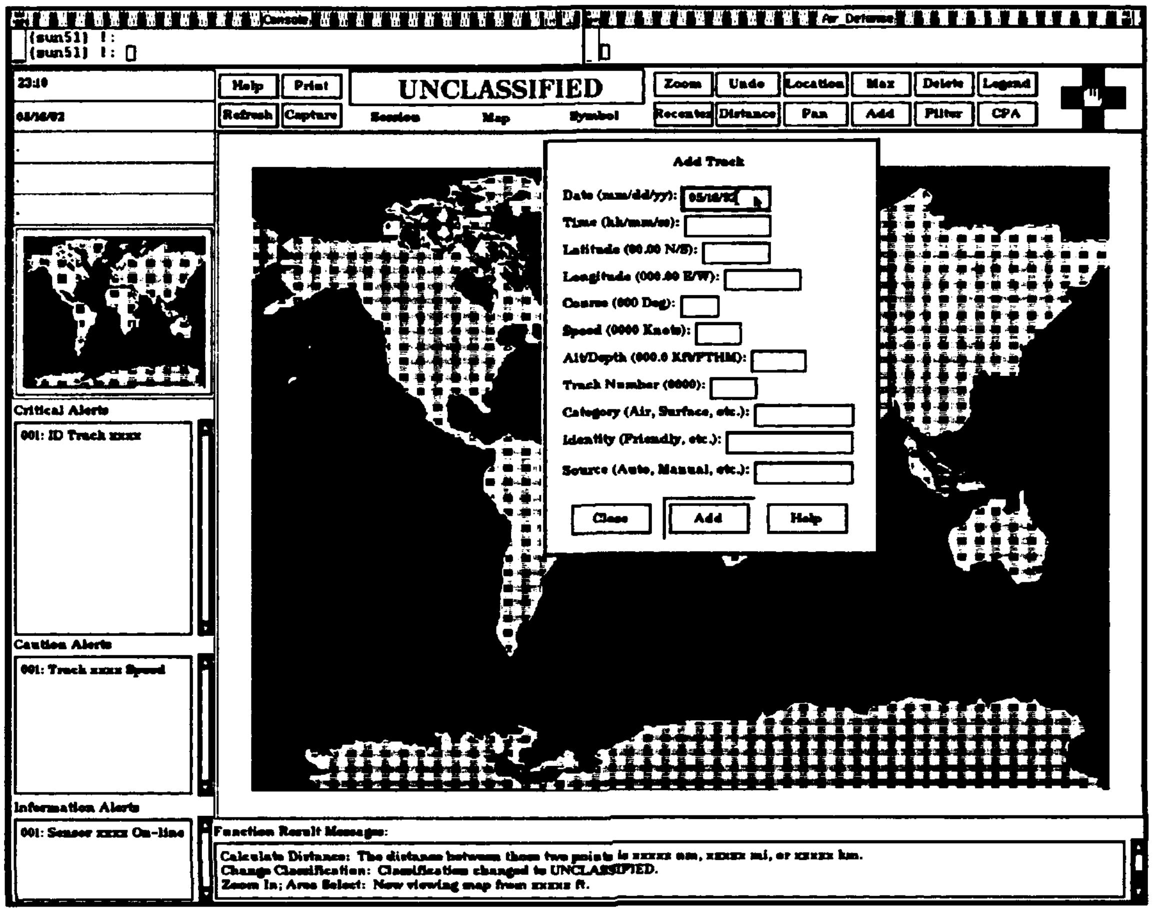
The Delete Symbol function allows the user to permanently delete a symbol from the system, whether automatically or manually added to the display. To delete an automatically added symbol, such as a track, the operation is confirmed with the client application software to ensure the symbol is not critical for some reason unknown to the user. No storyboard is provided for this function since, to execute it, the user needs only to select the function, double-click on the symbol, and acknowledge a confirmation panel. If the application will not allow the deletion, the user is notified. When the symbol has been deleted, it is no longer displayed on the map.

The Filter Symbols function lets the user select categories of symbols that need to be removed or added back to the display to help reduce clutter. Figure 32 shows the storyboard for this function.

The function, Edit Symbol Information, allows the user to change the graphical or textual information of a symbol. This is the same Edit function that can be invoked from the Show Symbol Information panel. Figure 33 shows the storyboard for this function.

The Show Symbol Legend function displays static information about symbols that could be displayed on the map. This helps the user verify what certain symbols represent. Figure 34 shows an example legend of track symbols. The legend panel has to be modified to add symbology. It was considered unnecessary to provide a function to allow the client application to add symbols to the legend.

The last symbol function is Calculate Closest Point of Approach (CPA). This function allows the user to display, given current information such as heading and speed, a predicted minimum distance between two symbols—one of which is presumed to be moving. The user invokes this function is a way similar to calculating the distance between two points. Selecting this function first causes the pointer to change shape. The first symbol is picked by single-clicking the left mouse button. If a new first symbol is wanted, a single-click of the right mouse button will deselect the first symbol. The user can then select another symbol without



```
[sun51] 1:
[sun51] 1:
```

23:10

05/16/92

Add Track

Date (mm/dd/yy): 05/16/92

Time (hh/mm/ss):

Latitude (00.00 N/S):

Longitude (000.00 E/W):

Course (000 Deg):

Speed (0000 Knots):

Alt/Depth (000.0 Kft/FTHM):

Track Number (0000):

Category (Air, Surface, etc.):

Identity (Friendly, etc.):

Source (Auto, Manual, etc.):

001: ID Track xxxx

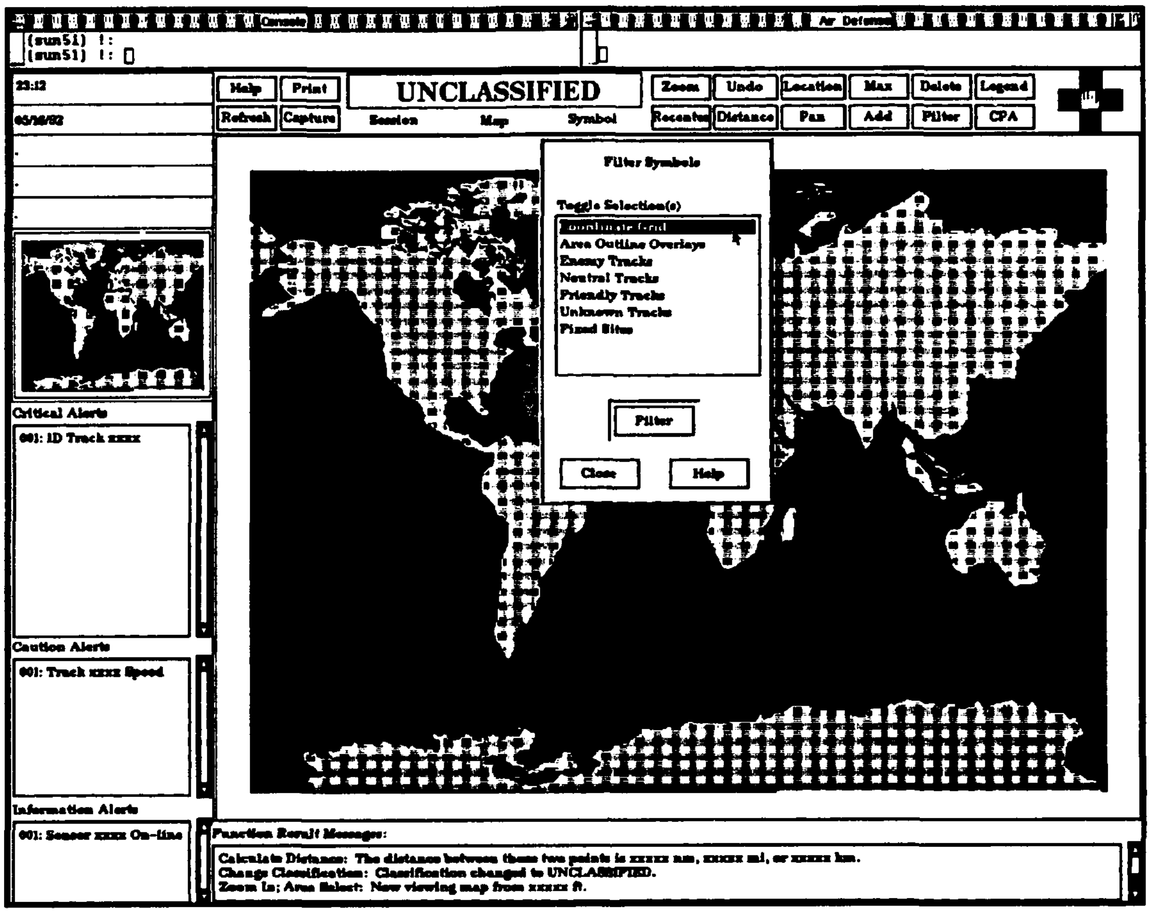
001: Track xxxx Speed

001: Sensor xxxx On-line

```
Calculate Distance: The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: New viewing map from xxxxx ft.
```

1. The Add Track panel prompts the user for manually adding tracks. This panel is displayed after the user selects the Track entry from the initial panel that lists overlays, fixed, and track symbols.
2. To get this Add Track panel, the user must select the Add Symbol function and double-click on the Track entry in the initial panel list. The initial panel and the Add Track panel allow the user to close the respective panel and cancel the function.

Figure 31. Add Track



001: ID Track xxxx

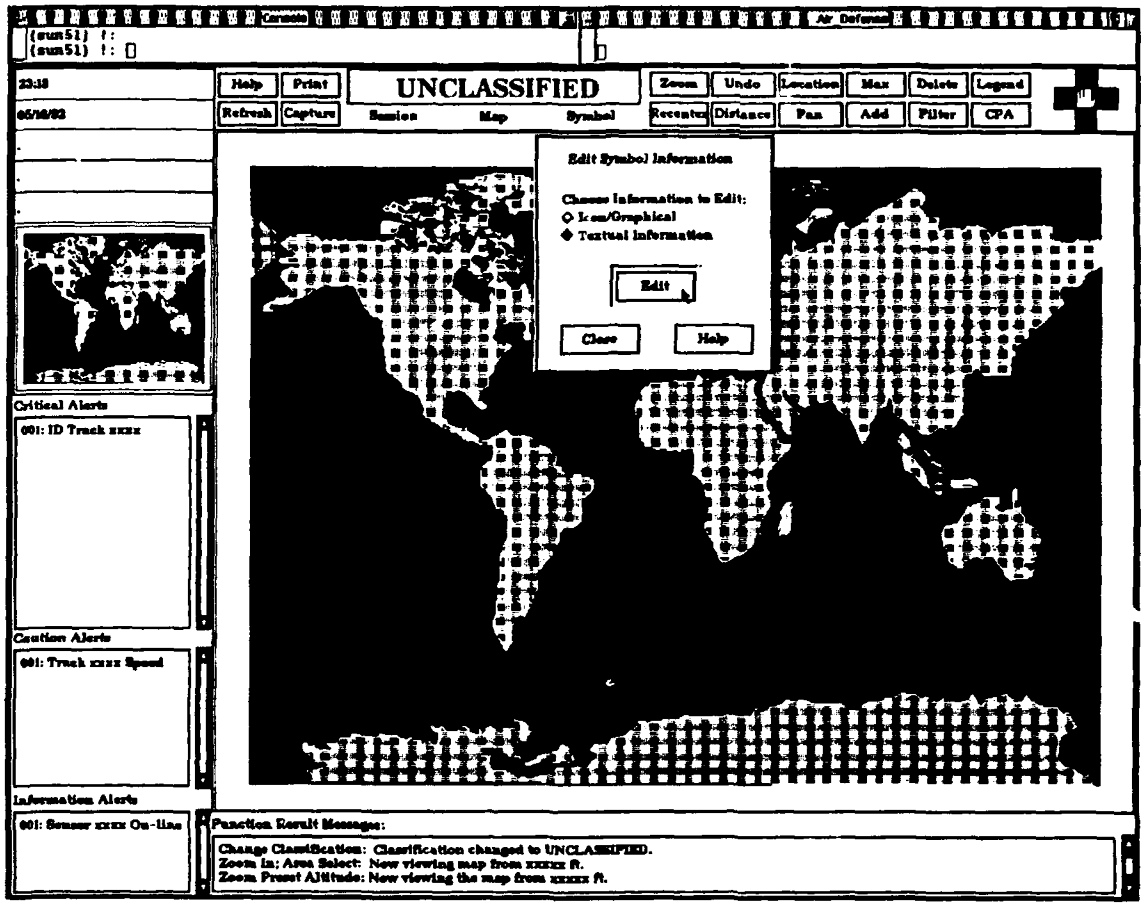
001: Track xxxx Speed

001: Sensor xxxx On-line

Calculate Distance: The distance between these two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The Filter Symbols panel prompts the user to select a category of symbols to remove or redisplay if already removed.
2. The user can select a category by either double-clicking a list entry or single-clicking and pressing the Filter button. The panel then disappears and any symbols being displayed that fit the selected filter category are temporarily removed. The user may invoke the function again and reselect the same category to redisplay the symbols.
3. Pressing the Close button causes the panel to disappear.

Figure 32. Filter Symbols



001: ID Track xxxx

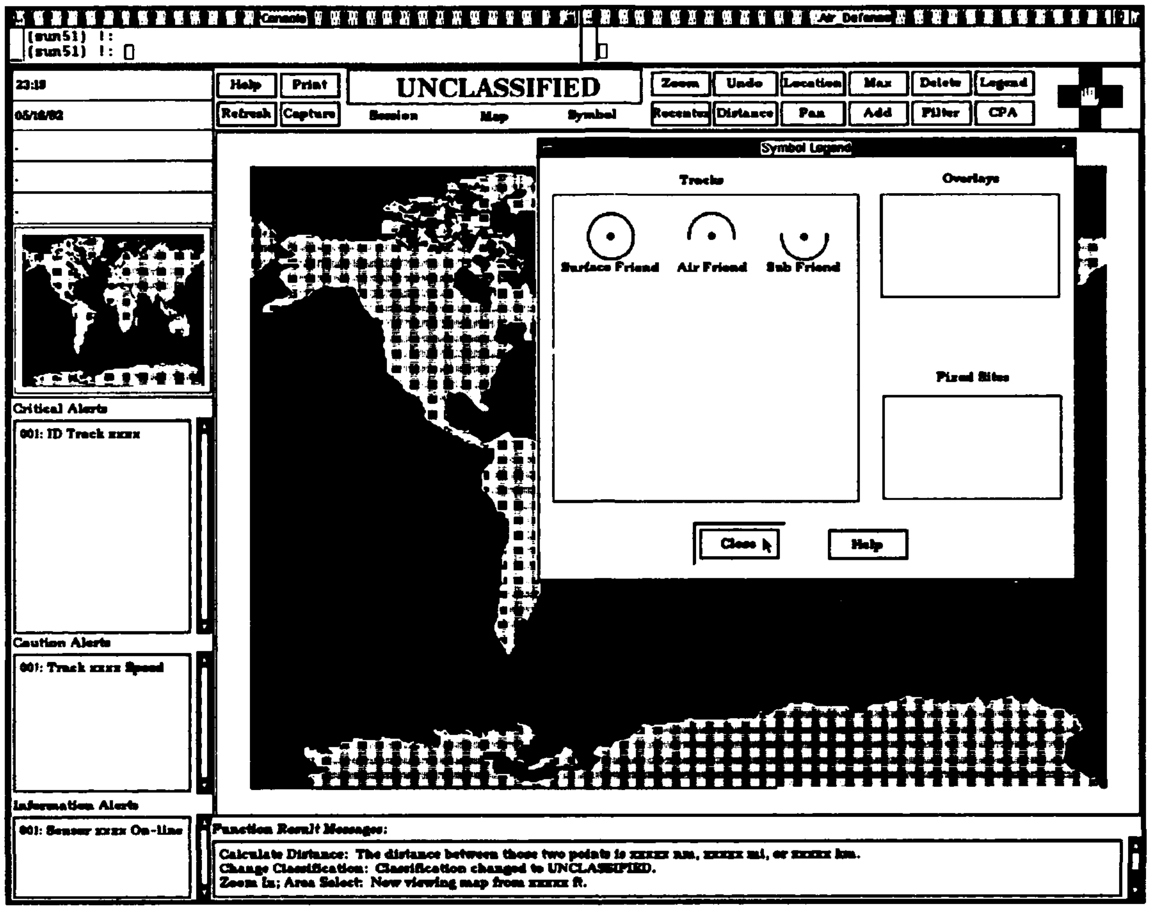
001: Track xxxx Speed

001: Sensor xxxx On-line

Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.
Zoom Preset Altitude: Now viewing the map from xxxxx ft.

1. The Edit Symbol Information panel lets the user choose either to edit the symbol icon with a graphical editor or to edit the textual information with a data entry panel.
2. The radio buttons prevent both options from being selected at the same time. The Textual information is the default selection.
3. Pressing the Edit button causes the appropriate panel to appear.
4. Pressing the Close button causes the edit panel to disappear and cancels the function.

Figure 33. Edit Symbol Information



001: ID Track xxxx

001: Track xxxx Speed

001: Sensor xxxx On-line

Calculate Distance: The distance between those two points is xxxxx nm, xxxxx mi, or xxxxx km.
Change Classification: Classification changed to UNCLASSIFIED.
Zoom In; Area Select: Now viewing map from xxxxx ft.

1. The Symbol Legend panel shows the three basic set of symbols: tracks, overlays, and fixed sites.
2. The user can select this function from the symbol menu or by using the Legend shortcut button.
3. Depending on the application, this user interface panel needs to be edited to show the applicable symbology.
4. Pressing the Close button causes the panel to disappear.

Figure 34. Show Symbol Legend

invoking the CPA function again. If the user wants to cancel the whole function. a double-click of the right mouse button must be done. Once the first symbol has been selected, the second symbol is selected in the same way by single-clicking the left mouse button. After the second symbol has been chosen, the CPA distance is calculated and the result is displayed in the Function Result list. This function can also be invoked by pressing the shortcut button.

5. Analysis Summary

Overall, storyboard prototyping provided a method to guide the use of the TAE Plus tool in documenting and changing the software requirements for the thesis software. Being able to rehearse some of the requirements was a great benefit of using a tool like TAE Plus. They were no longer just a list of features to include in the software, but instead were in a form that could be manipulated and interactively evaluated. While some of the original requirements were modified by going through the storyboarding process, no major changes were made.

a. Requirements Observations

The most significant observation was that the common functions needed to be divided into user-interactive and *application-callable categories*. It made sense to place some of the functions into both categories. However, with others it did not. Even some of the functions placed in both categories required different kinds of information based on whether they were invoked by the user or programmatically by the application. This will be explained in more detail in the design section.

It was also determined that the alert messages should be divided into three categories—critical, caution, and information. This was suggested by a student who had some experience with an airborne radar system that did not give a relative importance for a given alert message. Another observation led to creating five user definable labels in the upper-left area of the interface where previously there had been fixed labels for the date, time, etc. These labels now allow application to put whatever information is wanted in that area.

b. *Transition to Design*

In one sense the storyboarding process also became a user interface design exercise once the requirements were documented. However, this was not a disadvantage. The transition to the design phase was eased by using storyboard prototyping in this way.

Because of the ability of TAE Plus to run and create source code on the target workstation, preparing for the design and implementation phases was much easier to do than if the prototype had been created on another kind of system. In addition, the prototyping process showed that the software seemed feasible to create. Although this was not a major concern at the outset, determining whether or not the software could be written to meet the requirements was something to consider.

c. *Problems and Issues*

This leads to the fact that, in general, there are some disadvantages with storyboard prototyping. Andriole mentions, and as pointed out above, if the storyboarding software runs on a system other than on the one the final software will run, conversion problems may occur. Either the target system will not support certain features of the prototype system or the amount of effort for conversion will be too large to justify. [8, p. 68] Fortunately, this project did not encounter either problem.

However, other issues did arise. One issue was the significant time it took to learn TAE Plus. Although it has an easy-to-use graphical user interface, TAE Plus has many capabilities, and it took time to learn how to use these features effectively. Tips from experienced users helped the most when learning the finer details of using TAE Plus.

It should also be mentioned that this analysis method was not followed in a rigid sequence. Some tangent paths were taken and side discussions led to changes before formal evaluations were done. In retrospect, more formal and thorough critique sessions might have helped shorten the analysis, although it is difficult to estimate how much time it would have taken to setup more formal evaluations. In general, though, the prototyping method seemed to work well for the purposes of this project.

B. DESIGN

This section describes the design of the software for the map and symbol manipulation software. The main goal for the design was to create a loosely coupled, modular structure of Ada packages and to hide the implementation details as much as possible. The design method or approach used to create this structure could be classified as object-based and is documented with Buhr diagrams [14]. In this thesis the term "object-based" means that data and their related operations were encapsulated within the same Ada packages as much as possible. The "object" in this sense is the combination of the data and the specific operations related to that data. This means that most of the data is only accessible by external packages through calls to functions and procedures that either set or get the respective object data.

The software structure is object-based rather than object-oriented since no inheritance mechanism was used. The use of inheritance is usually considered necessary to call a design method or programming language object-oriented [39]. The exception to this was the object-oriented design of the track portion of the symbol software. The air track symbols were designed and implemented using object-oriented techniques to see how compatible inheritance and other object-oriented concepts would be with the object-based software. This is described in more detail later in this design section.

The design process included generating and restructuring the user interface source code, generating design diagrams, modifying the design diagrams, and regenerating the source code. This process was chosen in order to take advantage of some automated tools available for design. The TAE Plus workbench tool was used to generate the source code and the ObjectMaker tool was used to generate and edit design diagrams and also to regenerate some of the source code.

The design method and process were built on a software foundation of increasing levels of abstraction. These levels or layers of software included libraries and toolkits that hid the detail of the system hardware and software from the thesis software. The following sections discuss these layers and the design process.

1. The Software Design Foundation

The software foundation layers used for the software in this thesis are shown in Figure 35. At the lowest layer, were the operating system (OS) libraries and programs. The

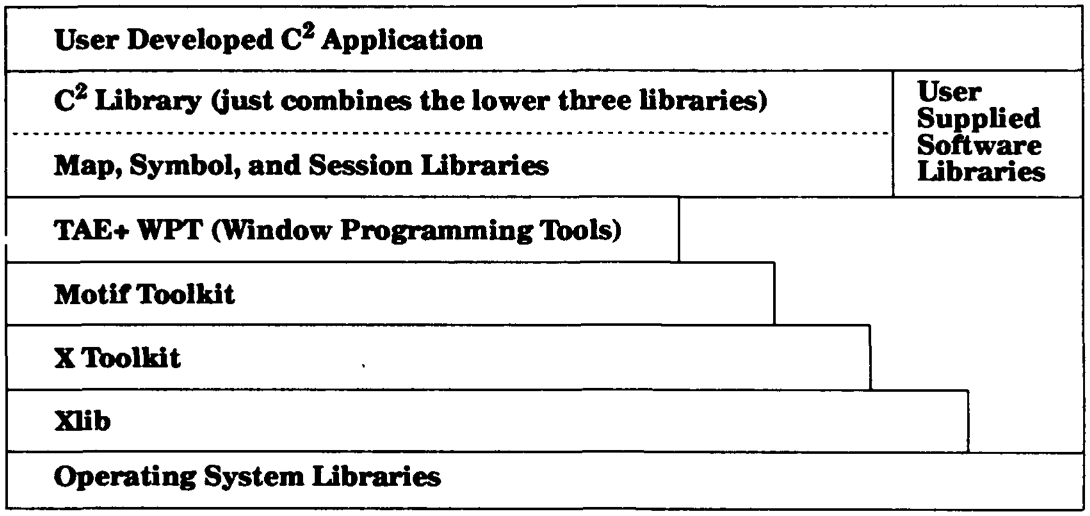

User Developed C^2 Application
C^2 Library (just combines the lower three libraries)
Map, Symbol, and Session Libraries
User Supplied Software Libraries
TAE+ WPT (Window Programming Tools)
Motif Toolkit
X Toolkit
Xlib
Operating System Libraries

Figure 35. The Software Layers

next layer was the X Library (Xlib) which included the functions that implemented the X protocol and map drawing functions. The third layer was the X Toolkit. This library included functions to create and manipulate windows with X using a higher level of abstraction than Xlib. The fourth layer was the Motif Toolkit which included functions to create windows that have a Motif look-and-feel. This is also known as a Widget Library since user interface items are created by manipulating widgets such as scrollbars, buttons, etc.

The fifth layer was provided by a library of TAE Plus functions. These functions allowed interaction with a pre-defined, limited number of Motif widget-combinations, called items, which were created with the TAE Plus workbench tool. The tool provided an interactive way to create items without programming. The items were stored in a resource file which was later read and displayed as the user interface. To allow access to the resource file and the displayed interface, TAE Plus provided a window programming tool (WPT) library. With WPT, the user interface display items and the events generated by user interaction with the items were managed.

The next layer is what the thesis software addressed. The map, symbol, and session libraries provide a C^2 application programmer with a domain-specific set of functions to create a display that shows the map and symbols specified by the application. The map and symbol libraries contain the map and symbol functions. The session library contains the related functions such as displaying alert messages and information in the user-definable labels.

These libraries are separated from the C^2 library by a dashed line to indicate that the C^2 library merely contains the specifications of the other libraries. In this way the client application is coupled to only one library instead of three.

This layer also can contain user-supplied software libraries that can be anything an application programmer may need for a specific application. As Figure 35 shows, the three C^2 libraries and the user libraries potentially have access to the lower software levels more detailed functions are needed than the higher layer libraries can provide. To prevent problems with displaying the user interface, it is not advisable for the user libraries to access any library other than the OS libraries. The reason for this advice has to do with the way graphics are drawn using X and will become more clear later on in the design discussion.

The last (top) layer contains the client application program that can call the application library functions. A programmer has to add the specific application functionality that interacts with the map user interface. Client application functions could include such things as receiving sensor information, managing databases, and other such operations.

In summary, these software layers provide increasing levels of abstraction that hide the underlying programming details. Thus, more effort can be put into what the application should do and less effort on how the application should implement the user interface.

2. Generating and Restructuring the User Interface Source Code

The design process began with automatically generating the user interface source code after the storyboards were developed. The code was generated with the TAE Plus Ada code generator. This tool created multiple Ada files and packages which could also be compiled into an executable program for the user interface. The code generator needed about 15 elapsed seconds of processing time on a lightly loaded Sun SparcStation 2 to place over 13400 source lines of code (SLOC), including blank lines and extensive comments, into 48 files. The code generator could have put the source code into one file. However, because of the significant size of the user interface code and the time-saving advantages of selectively compiling separate Ada files during later development, the multiple file option was used.

Even though TAE Plus generated multiple packages in multiple files, code restructuring was needed to let a client application and a user interact with the interface. The restructuring involved creating a monitor task to control user and application access to the user interface. Source code templates (Ada package skeletons with form but no function) of the session, map, and symbol library packages were also created. In addition, a sample client application was created to demonstrate the use of the C^2 libraries. After the restructuring, the whole set of software packages was ready to run through the ObjectMaker reverse-engineering product.

As a note of clarification, the code restructuring and code template creation did not have to be done before generating the design diagrams in the next step. At the time, it seemed best to do some initial code restructuring and to manually create some of the templates in order to gain a better understanding of what was required to integrate additional packages with the generated code. To emphasize that the manual template creation could be excluded, the next two sections will describe the design of the modified TAE Plus generated code as though no initial templates were created.

3. Generating the Design Diagrams

The next step in the design process was to generate the design diagrams. The ObjectMaker tool was used to process the TAE Plus generated Ada code into Buhr design diagrams. This process is also called reverse-engineering because typical software engineering involves creating source code from design diagrams—not the reverse. This section describes the design diagram notation, the reverse-engineering process, and then the design of the TAE Plus source code before any restructuring or code template additions.

The Buhr style of design diagrams was chosen because it supported Ada language concepts with a concise set of notation and was supported by the ObjectMaker tool. The original Buhr diagram notation is fully described in the book *System Design with Ada* by R J. A. Buhr [14].

Figure 36 shows a legend of the basic Buhr design symbols used in the thesis software design. In Buhr notation a package is represented by a box with rounded corners as shown by Packages A and B. An abstract data type and subtype are represented by ovals with a dashed border. An object (or variable) is represented as an oval with a solid border. A subprogram, such as a procedure or function, is shown as a rectangle. A task is represented by a parallelogram. Select statement blocks and entry/accept sockets are depicted as smaller parallelograms. Determining what a specific parallelogram represents depends on the context

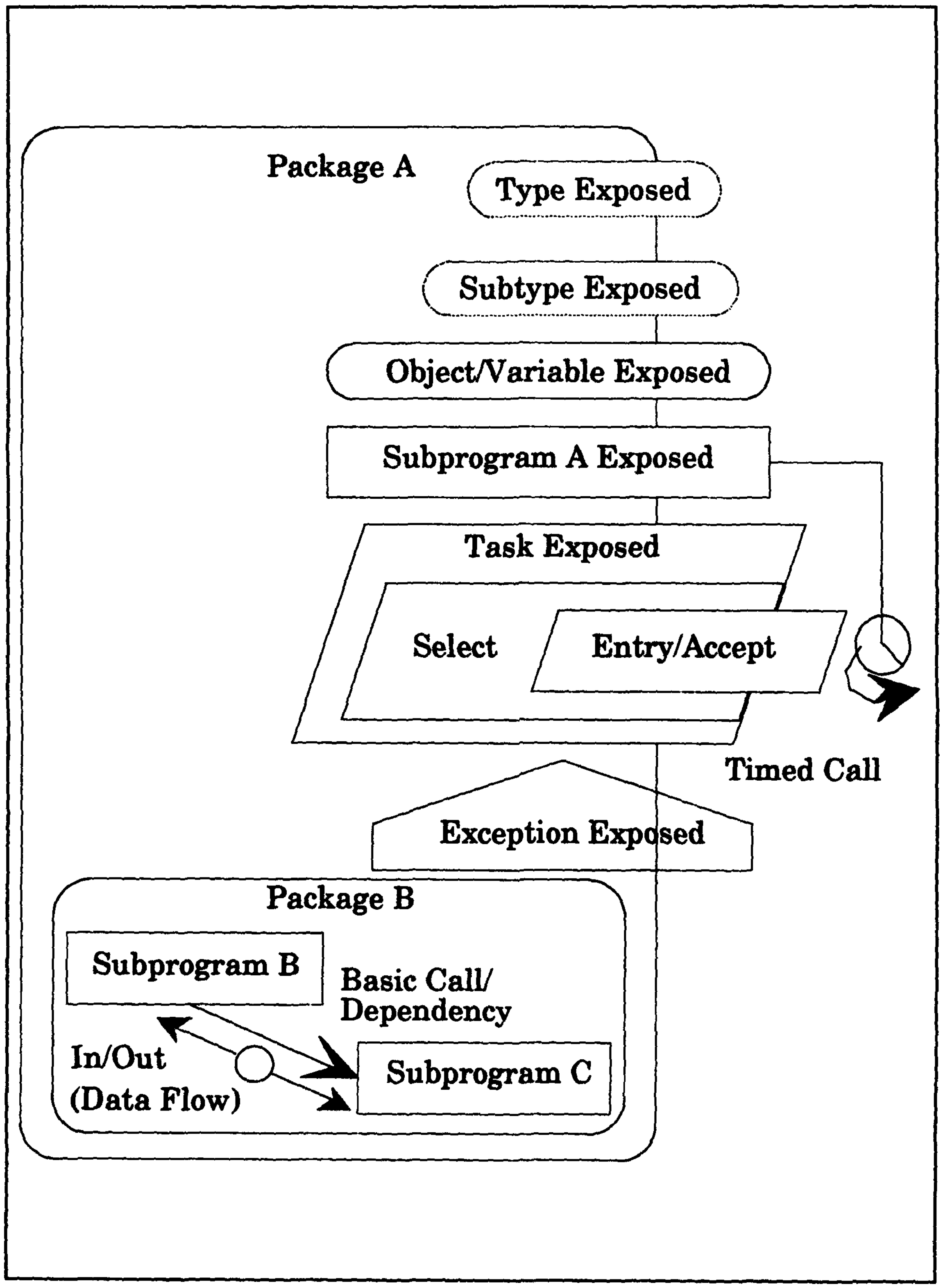


Figure 36. The Buhr Diagram Legend

of its use. In ObjectMaker the usage context is automatically managed. The notation for a timed entry call is shown as an arc with a clock symbol next to the arrowhead. A basic call for an entry or subprogram is represented by a regular arc as shown in Package B. An arc can also indicate a package dependency when the arrowhead does not point to a callable procedure. Data flow is shown by small bubble arcs placed next to the call arcs. An Ada exception is represented by an irregular pentagon. All of the symbols labeled as "exposed" are considered visible to outside program units that access the package. This means their definitions are list in the packaged specifications. The Package B and its Subprograms, B and C, are within Package A and are hidden from external units. Other Ada constructs such as generic packages and conditional calls also have symbols in the Buhr notation, but they are not shown since they are not used in the software design.

The reverse-engineering procedure using ObjectMaker actually involved two separate steps: generation of compilation dependency diagrams, followed by generation of the Buhr diagrams. The first time the TAE Plus generated source code was processed, ObjectMaker created several files that graphically showed how the Ada packages depended on each other for compilation. To create the dependency diagrams, ObjectMaker, on a lightly loaded system, took three to five elapsed minutes to process the 13413 SLOC. These diagrams were actually a subset of the Buhr diagrams. Any package that could not be found (e.g. an external library like TAE Plus) was represented by a cloud. In addition to the graphics files, a text file was created that contained the list of names of the Ada source code files in order of compilation. This file was needed for the second step.

With the ordered file list, ObjectMaker could then generate the Buhr diagrams. Depending on the system load, Buhr diagram generation took 15 to 20 elapsed minutes. The result was about 70 files that contained Buhr diagrams for each of the packages and subprograms and a few diagrams showing general package and subprogram relationships.

Figure 37 shows a general Buhr diagram of how the TAE generated multi-file Ada source code was structured. This figure is a condensed version of the original diagram that was actually generated. The original diagram was too detailed to print at a reasonable scale. Still, this diagram shows the major pieces of a TAE Plus generated program. In this figure, the application is called C2 Map. An event loop in the main procedure handles the initial event processing and on each pass through the loop checks a global variable to see if it should

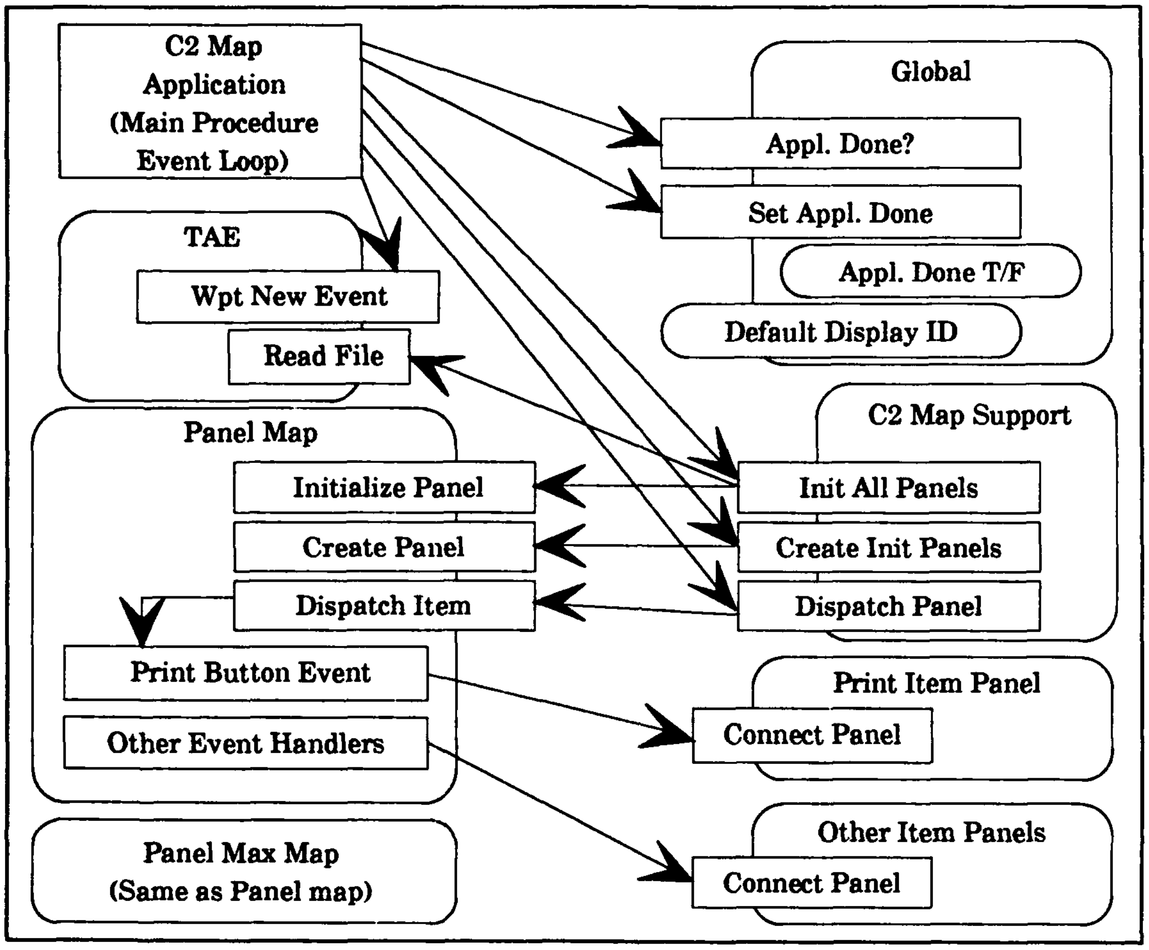


Figure 37. The Original TAE Plus Generated Source Code

terminate. The loop itself may set the global variable or, more likely, one of the event handler procedures would set the variable after a quit button was pressed.

The main procedure first displays the initial user interface, called the main panel (or panels) by calling the Init All Panels and Create Init Panels procedures in the C2 Map Support package. The main procedure then calls the TAE Wpt New Event function to get ready for an event. Each time the user causes an event to occur, such as by pushing the Print Button on the main panel, the main procedure calls the Dispatch Panel procedure in the C2 Map Support Package. This procedure determines from which main panel the event was created and calls the Dispatch Item procedure of that panel. The Panel Map package in the figure represents a main panel that contains items that may create events. The Panel Map Dispatch Item procedure then determines which specific item created the event and calls an event handler procedure to process the event. In the case of the Print Button Event, yet another panel is needed to prompt the user for printer and file information. Therefore, the Connect Panel procedure is called in the Print Item Panel Package. This package contains some more event handler procedures to process an item event created from that panel, such as by pressing an execute button. The Other Event Handlers may or may not need to connect to another panel. If not, an event handler inside of the Panel Map package executes the necessary statements. The Panel Max Map package represents another main panel and contains almost the identical procedures as the Panel Map package.This whole series of procedure calls can be a little confusing at times, but the hierarchical structure helps quickly pin-point the event-creating item.

Overall, ObjectMaker does a pretty good job of automatically generating Buhr design diagrams from source code. Although only one general diagram was shown, each package and subprogram has its own subdiagram. With these diagrams, the structure of automatically generated TAE Ada code was easily analyzed. Although the diagrams proved useful for software analysis, modifying them to add functionality proved difficult because of the small figure scale and busy organization. It seemed easier to create new diagrams from scratch. The next section explains how diagrams were created to describe the design of the application functionality behind the user interface.

4. Modifying the Design Diagrams

After the Buhr diagrams were generated, the next step in the design process was to restructure the software package and subprogram relationships. This was done with the ObjectMaker tool using the graphic editor that allows manipulation of the Buhr diagrams. In reality, the design diagrams shown in the following figures were created from scratch since the effort to modify the automatically generated diagrams seemed too great. The generated diagrams proved useful for analyzing the TAE Plus generated software, but were too complex to easily modify for the full application design. This section describes how the design of the client application and user interface functionality was created using ObjectMaker.

a. The High-Level Modified Design

The high-level Buhr diagram of the modified design is shown in Figure 38. The arcs in this diagram show package dependencies rather than procedure calls in this case.

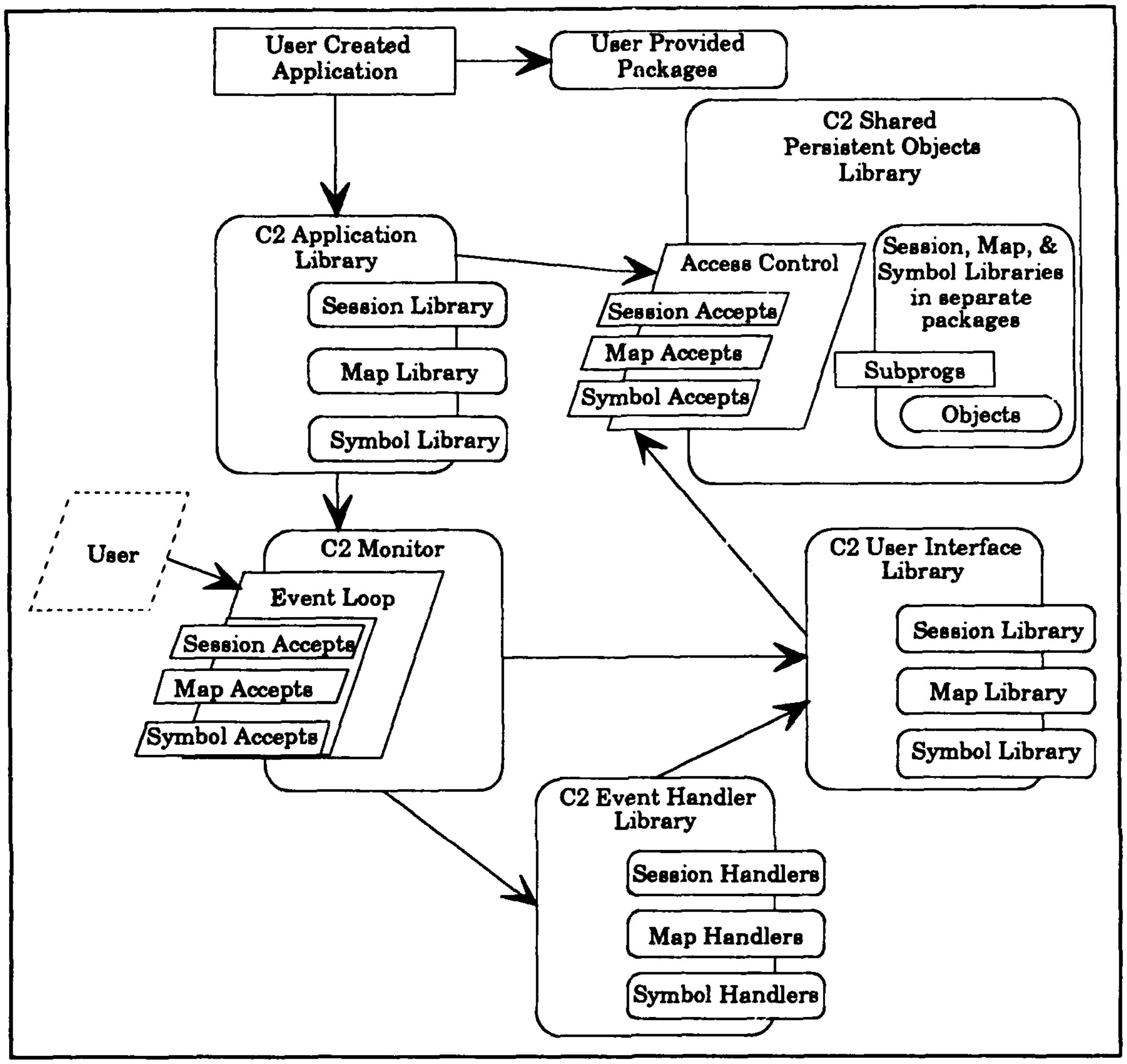


Figure 38. The Modified Design Diagram

Starting at the top of the diagram and continuing counter-clockwise, the User Created Application can access external resources with user provided packages. These external resources could include things such as networks or databases. The client application accesses the user interface through the C2 Application Library. The application library contains subprograms organized by session, map, and symbol functionality—a common structure throughout the software design. The subprograms access the user interface through entry calls to the C2 Monitor package Event Loop task. This task runs concurrently with the client application and prevents simultaneous client-application and user access to the user interface.

The application library subprograms also access shared data objects through the C2 Shared Persistent Objects Library. This library also has a task to prevent simultaneous updating of object data.

The C2 Monitor package is needed since the X-based user interface can not be interrupted in the middle of an operation. In other words X is not reentrant. For example, a user-generated event that causes a map redrawing operation can not be interrupted to handle an application-generated update of an information label. The monitor insures an operation that affects the user interface fully completes before the next operation is started.

The C2 Event Handler Library contains the consolidated event handlers that were initially created by the TAE Plus code generator and then modified to add functionality. Since the Main and Max map user interface functions create practically identical events, their separate event handlers were combined into a related package structure to reduce code redundancy.

The C2 User Interface Library handles all the updating of the user interface requested by subprogram calls initiated from the client application. The event handlers also call the user interface library functions to perform interface updates in response to user actions.

The C2 Shared Persistent Objects package is needed to maintain the objects that are currently part of the user interface. The object data is stored only in primary memory. Access to the data is handled by an Access Control task that manages updates and allows simultaneous reads.

The next section describes each of these packages in more detail.

b. *The Application Library*

The design of the C2 Application Library focused on the functions and data types that a client application would need in order to interact with the user interface and the associated data. As shown in Figure 39, the application library contains functions that update the user interface through the monitor and access the data in the object library. The directed arcs show what kinds of calls are made from the specific subprograms in the application library. The decision was made to allow some of the arcs to cross between different categories of functions in order to access certain object data. For example, the Init System "session"

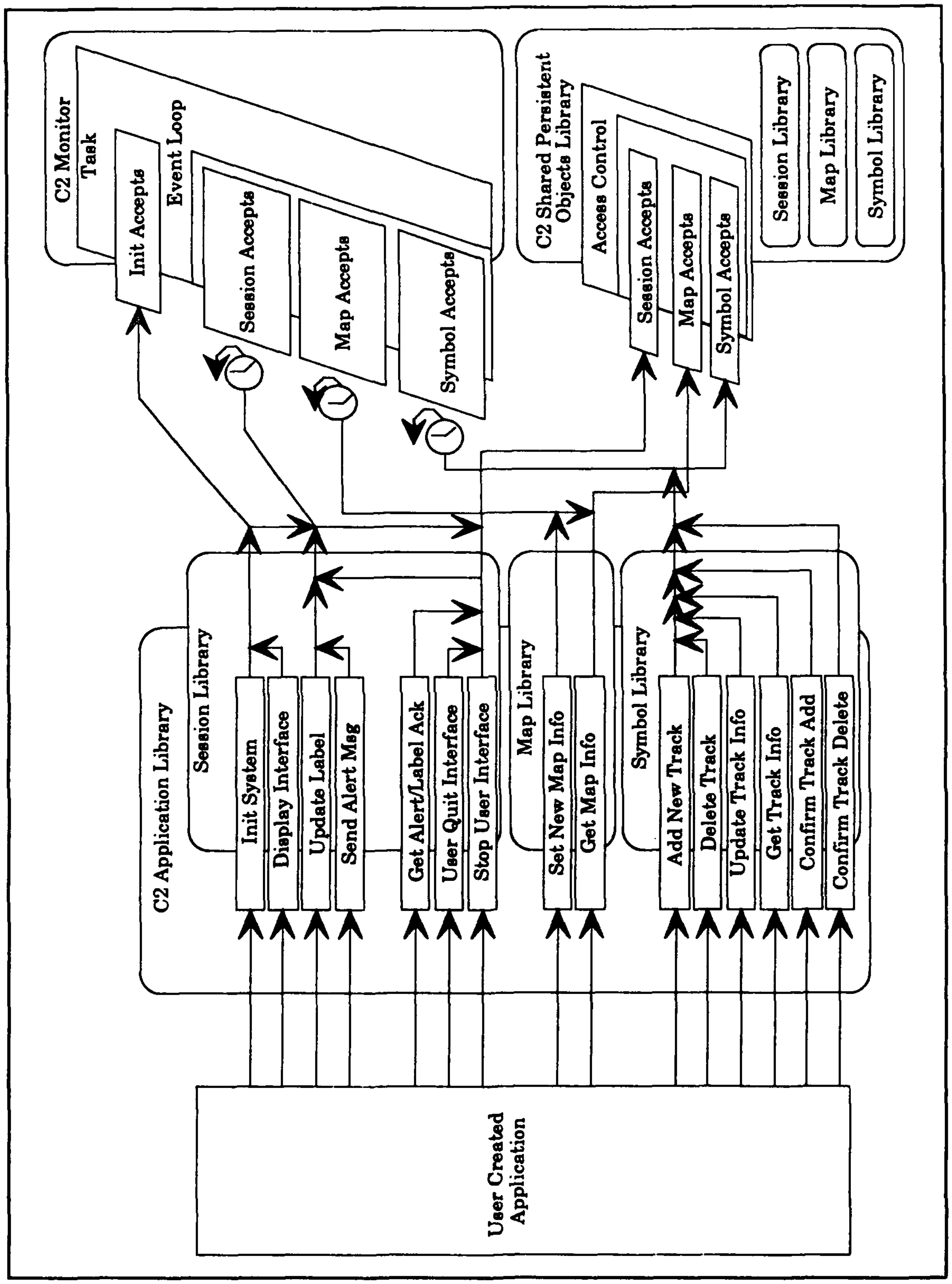


Figure 39. Application Library Subdiagram

function actually initializes the map information for the first map to display, although this is not shown with an arc in the diagram. When the application library subprograms are called, they first store or retrieve object data received from the client application and then make monitor entry calls to signal the user interface library to make any needed user interface display updates.

The design issue of how to make the shared data types visible to the client application proved challenging. It seemed that either a separate shared data type package could be provided for the client application and other packages that needed it or the shared object library could contain visible data types. Another alternative was to duplicate the essential types and do type conversions where necessary. The decision was made to make the types and associated constants visible to the client application through the application library. For specific information on what each function does and how the user interface accesses the shared data types, see Appendices A and B.

c. *The User Interface Monitor*

The C2 Monitor controls access to the user interface and contains a task that specifies the entries that the application library can use. Figure 40 shows the subdiagram for the monitor. In this diagram only the application routines that make monitor entry calls have arcs connected to the task. The unconnected subprograms and also most of the connected subprograms in the application library update object data in the shared object library. The monitor event loop insures that the user created events and the client application subprogram entry calls are handled one at a time without interruption. If a user event occurs, the event handlers in the C2 Event Handler Library are called. If an application entry call is made, the C2 User Interface Library is called directly. A minimum amount of data is passed through the monitor task. The user interface library is given enough key data to go and look up any other needed data in the shared object library. This way, subprograms and entry calls do not pass data that can be found in the shared library through the monitor. This also prevents the monitor package from needing access to the shared object library. For specific information on what each accept statement does, see Appendix B.

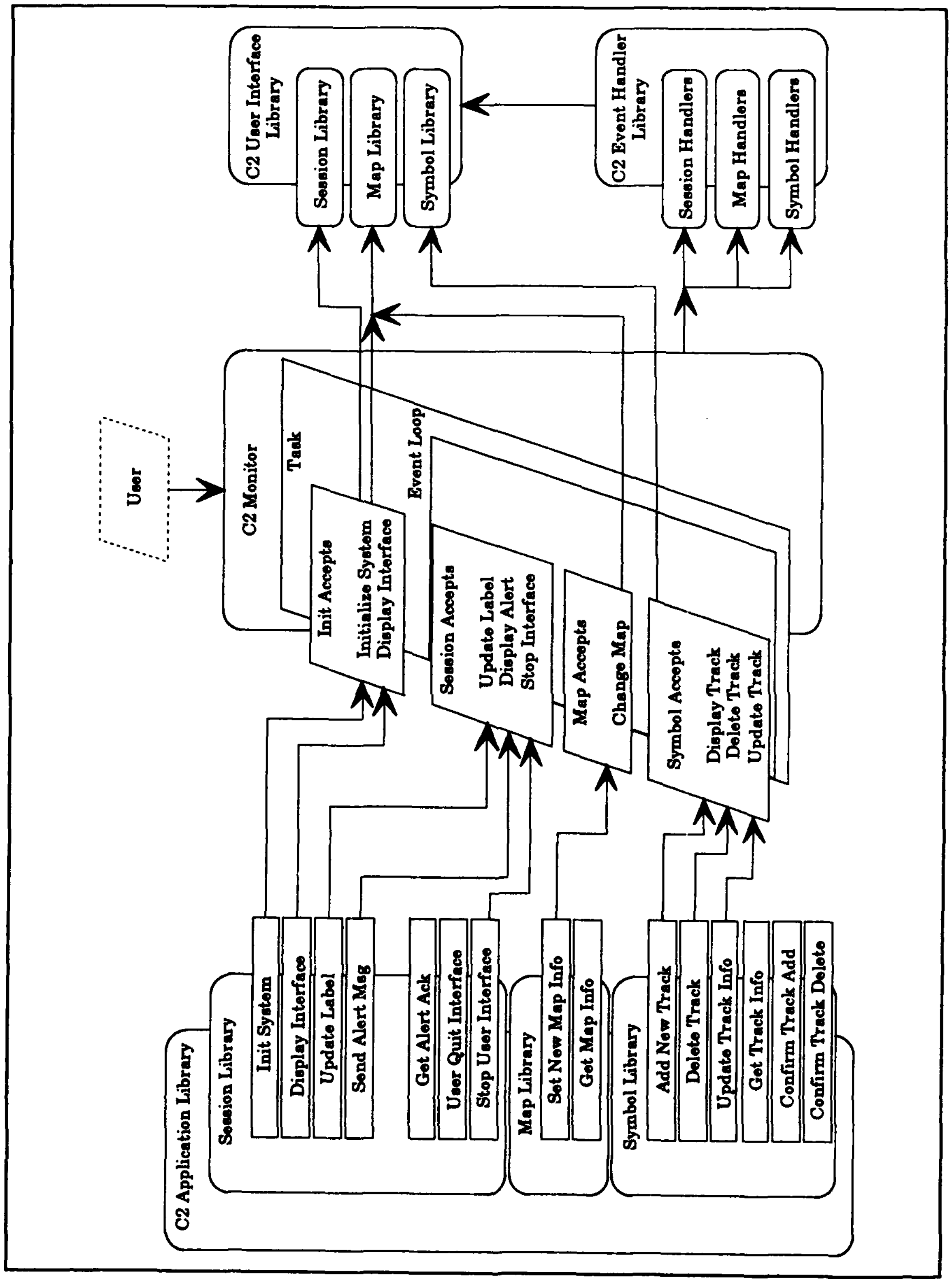


Figure 40. User Interface Monitor Subdiagram

d. The Event Handler Library

The purpose of the C2 Event Handler Library subprograms is to identify the item of the user interface that created the event and perform the appropriate function. Figure 41 shows the subdiagram for the event handler library. Each event handler subprogram corresponds to one of the items in the user interface. Some of the event handlers simply connect a button or menu item to another panel. This panel in-turn also has event handlers that correspond to the panel items. If an event handler needs to update something other than to display a connected panel on the user interface, the appropriate user interface library subprogram is called.

Even though the event handlers do not modify the user interface, they need access to the TAE Plus function library and the Xlib data types. With the TAE library the event handlers access information from the user interface resource file to find information about what item the user acted upon to cause an event. Since the TAE library uses a limited number of Xlib data types, the Xlib must also be accessible to the event handlers. In addition, the event handlers have access to system functions and programs through the system call library. These operations include functions such as refreshing the screen or saving a captured screen to secondary storage. Because these functions do not involve redrawing the user interface or shared data, they can be segregated in a system library which contains the non-portable Ada software. For more information about the event handler library, see Appendix B.

e. The User Interface Library

The purpose of the C2 User Interface Library is to update information on the user interface and draw the map and symbol graphics. Figure 42 shows the subdiagram for the user interface library. The client application library calls subprograms in this library through the monitor and the event handlers call the subprograms in response to user actions. To perform a function, the user interface library updates and/or gets data from the shared object library if needed and performs the requested user interface operation using the TAE Plus library, Xlib, or System Calls. The subprograms which are not visible outside of the user interface library are used by the visible subprograms to perform more abstract operations. The reason for the underline characters in the names of Figure 42 is so the diagram could be used

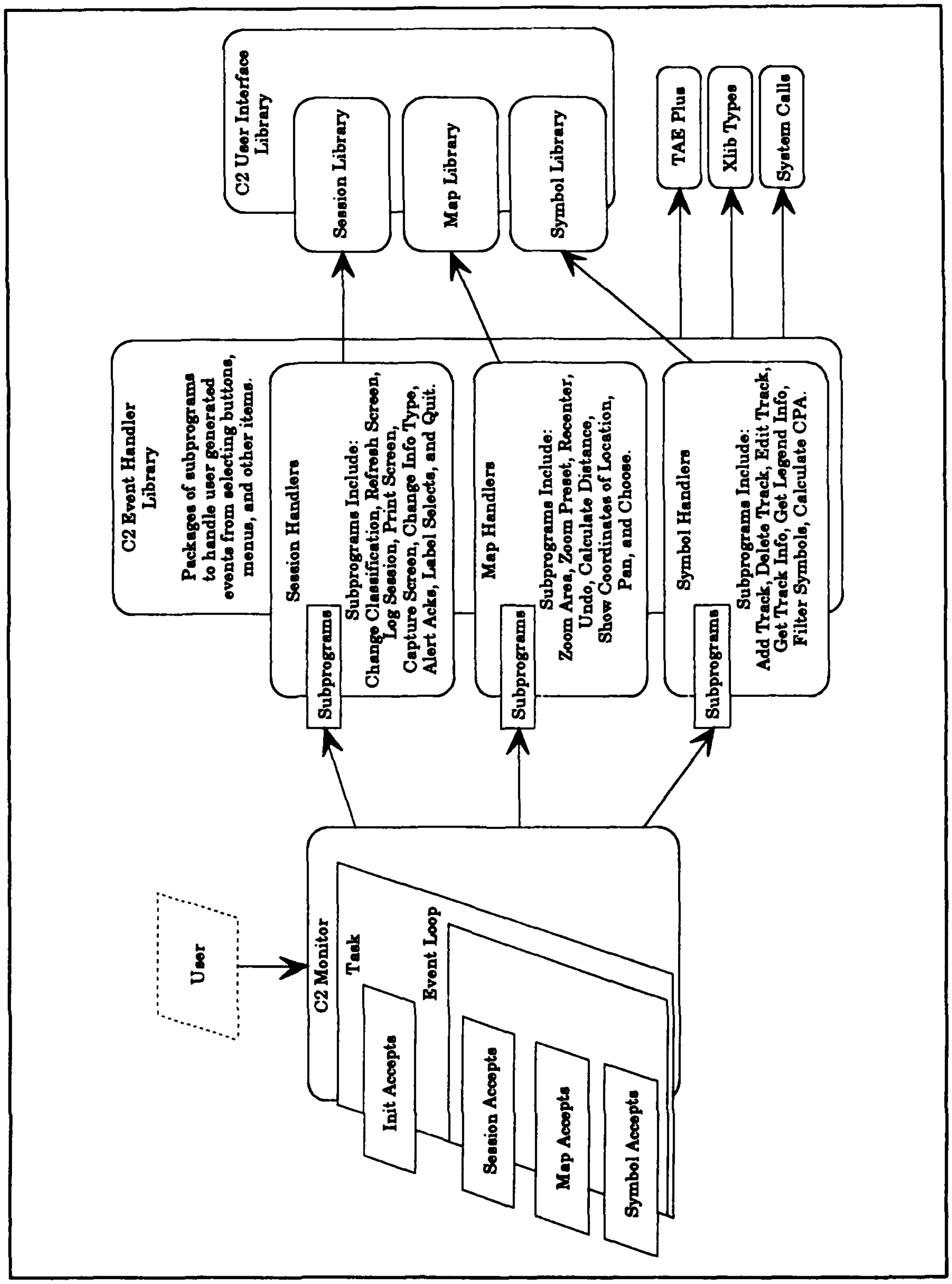

C2 User Interface Library
Session Library
Map Library
Symbol Library
TAE Plus
Xlib Types
System Calls
C2 Event Handler Library
Packages of subprograms to handle user generated events from selecting buttons, menus, and other items.
Session Handlers
Subprograms
Subprograms Include: Change Classification, Refresh Screen, Log Session, Print Screen, Capture Screen, Change Info Type, Alert Acks, Label Selects, and Quit.
Map Handlers
Subprograms
Subprograms Include: Zoom Area, Zoom Preset, Recenter, Undo, Calculate Distance, Show Coordinates of Location, Pan, and Choose.
Symbol Handlers
Subprograms
Subprograms Include: Add Track, Delete Track, Edit Track, Get Track Info, Get Legend Info, Filter Symbols, Calculate CPA.
User
C2 Monitor
Task
Init Accepts
Event Loop
Session Accepts
Map Accepts
Symbol Accepts

Figure 41. Event Handler Library Subdiagram

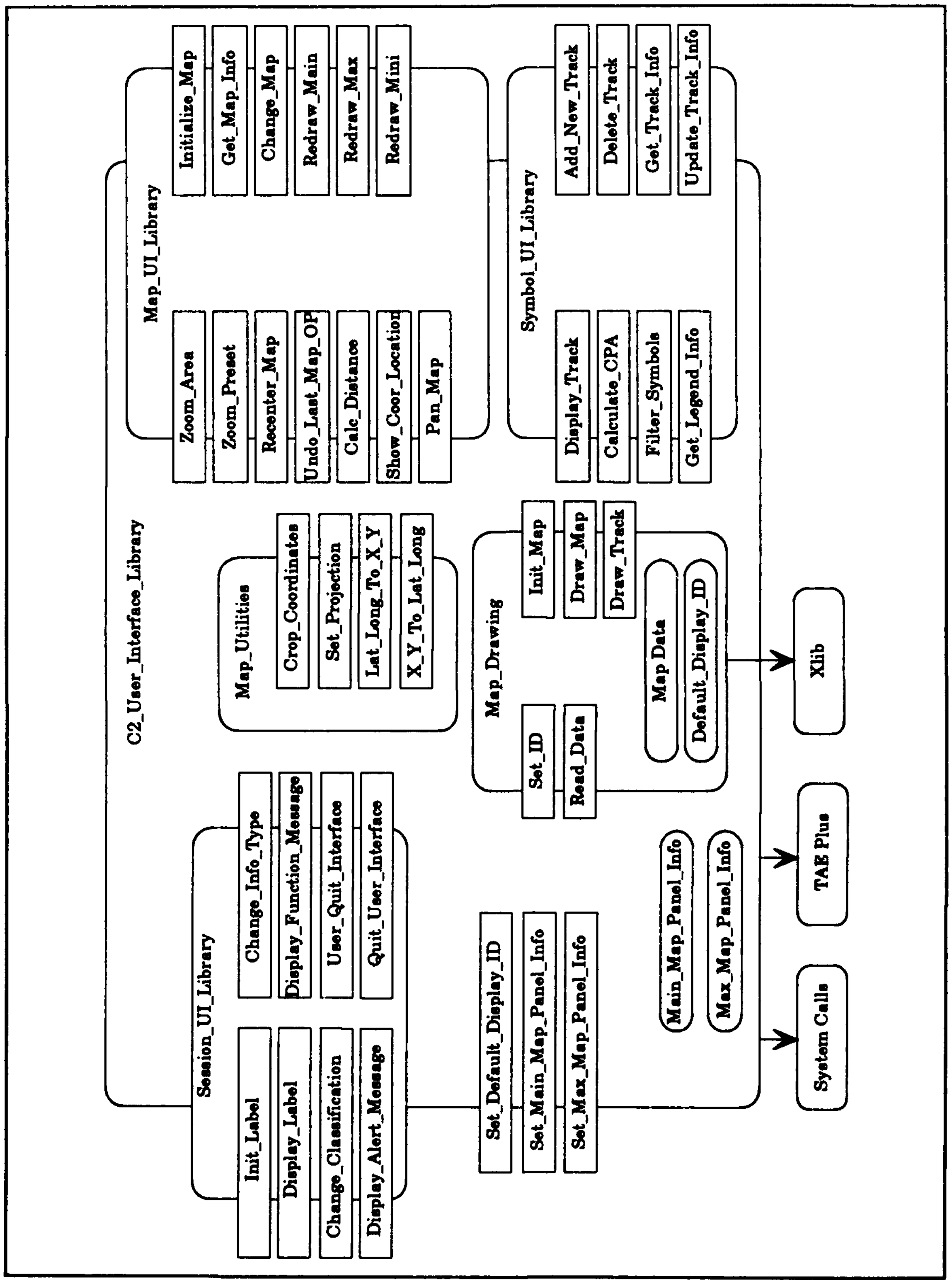


Figure 42. User Interface Library Subdiagram

to automatically regenerate Ada source code templates. The code regeneration is described in a later section.

The design of this library could have included a task that would routinely update user interface items such as tracks. As the design stands now, the client application must constantly give new position information and make a call to redisplay the track at the new location. The automatic approach was part of the work done by Stockwell in the LCCDS project [56]. The decision was made to leave out the automatic updating feature in this project. For more specific information about the user interface library, see Appendix B.

f. The Shared Objects Library

As mentioned in the previous paragraphs, the C2 Shared Persistent Objects Library contains the data, subprograms, and data types that are used by both the application and user interface libraries. Figure 43 shows the subdiagram for the shared persistent objects library. Access to all of the data is controlled by a task that prevents simultaneous changes and also changes while the data is being read. Concurrent reads are allowed. The data structures all have set and get subprograms that update and modify the data when called by another subprogram. The data types and associated constants are visible to allow their use by calling subprograms. For more detail on the shared objects package, see Appendix B. For the discussion of the object-oriented design portion of the track objects, see the last part of this design section.

5. Regenerating the Source Code

Once the design diagrams were drawn, ObjectMaker was able to automatically create Ada source code templates. However, for this thesis software, only the C2 User Interface Library Ada templates were created in this way. ObjectMaker quickly generated the code and the style was very readable. One good feature was that the code could be previewed from within ObjectMaker to make sure the names and code structure looked like what was expected. For example, it was learned while using the preview feature that the separate words making up diagram label names must have underscores between them or the words after the first one are ignored.

Although the code regeneration capability was nice to have, it was not as useful as first thought. After the templates of the user interface library packages were generated, not

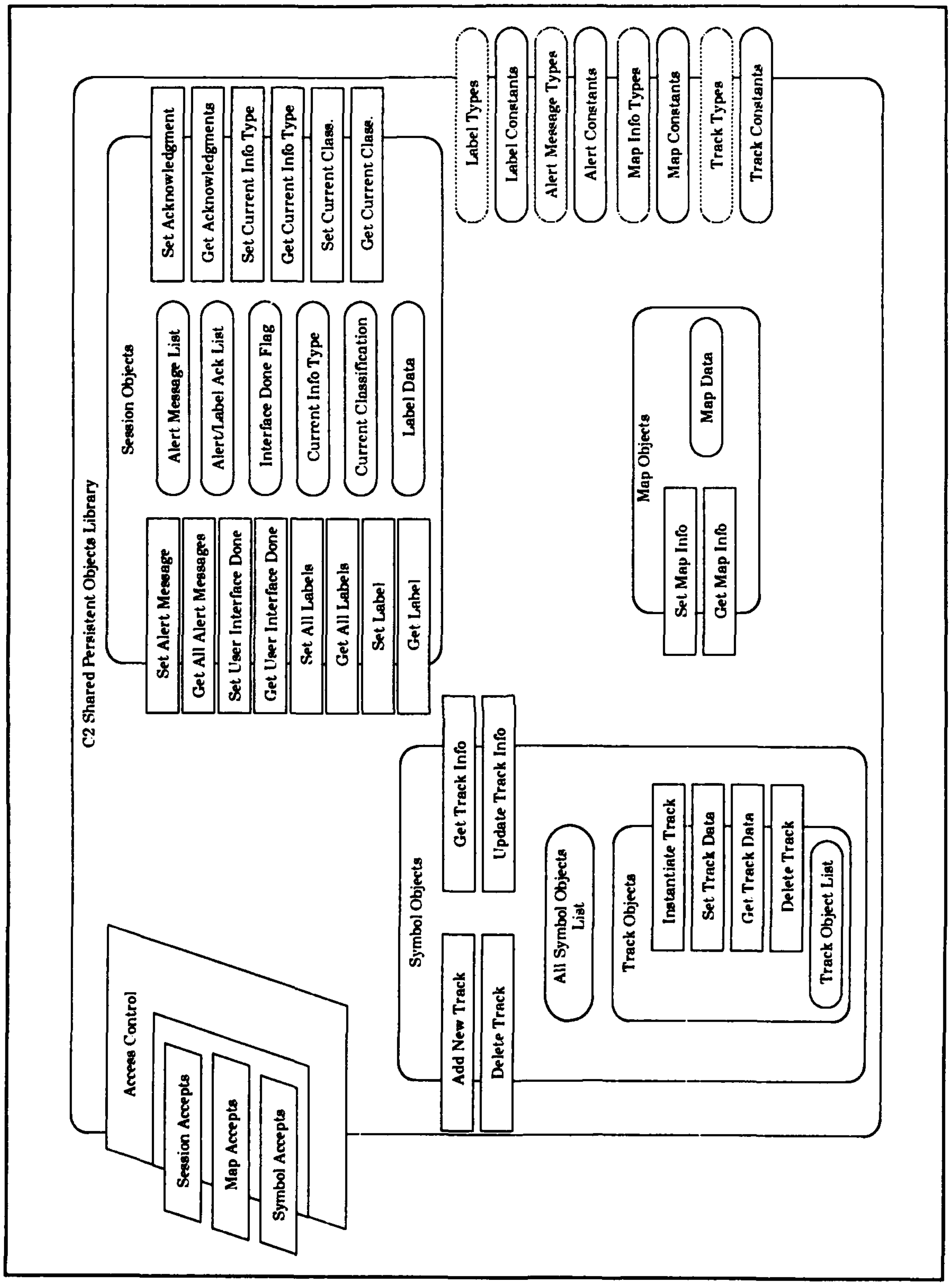


Figure 43. Shared Persistent Objects Library Subdiagram

surprisingly the design had to be modified after some of the implementation had been done. Changing the diagrams was easy enough, but regenerating the templates proved not to be useful since the implementation code had to be cut from the old design and pasted into the new design. Granted, diagrams of the old template code with the new application code could have been reverse-engineered, modified, and then the code regenerated. However, the new application source code comments would have been lost and, generally, the time involved would have exceeded the benefit. For small changes, separate and manual modifications to the design diagrams and code, to keep them consistent, were easier and faster. Using ObjectMaker to make major design modifications for a large project might be worthwhile. However, this project was not large enough to benefit in this way.

6. The Object-Oriented Track Design

The last part of the design to discuss is the object-oriented track portion of the shared objects symbol library. The decision was made to design this part of the software using common object-oriented concepts to see what level of effort was needed to integrate regular Ada software with an object-oriented extension of Ada called Classic-Ada [51]. The design was primarily based on design work done for an object-oriented track database for the LCCDS project [21].

The class hierarchy for the design of the air tracks is shown in Figure 44. The connected and solidly-lined boxes show the class inheritance relationships that define the air tracks. The dashed-lined boxes show some associated classes that do not have direct bearing on the definition of tracks. The detached classes are called class 'constants' and are used to return a constant value. These constants were considered a better alternative to making them class variables within the track class hierarchy [21, p. 209].

The class definitions that describe the variables and methods (operations) of the track-related classes are shown in Figure 45. The CLASS name corresponds to the names shown in the class hierarchy in Figure 44. Superclasses are the classes from which the current class inherits. Class Variables are those that the current class and any inheriting class shares. The values of class variables are shared among all related classes. In this design no class variables are used. The Instance Variables are inherited by subclasses but the values are not shared. Each object that is created from a class with instance variables maintains its own

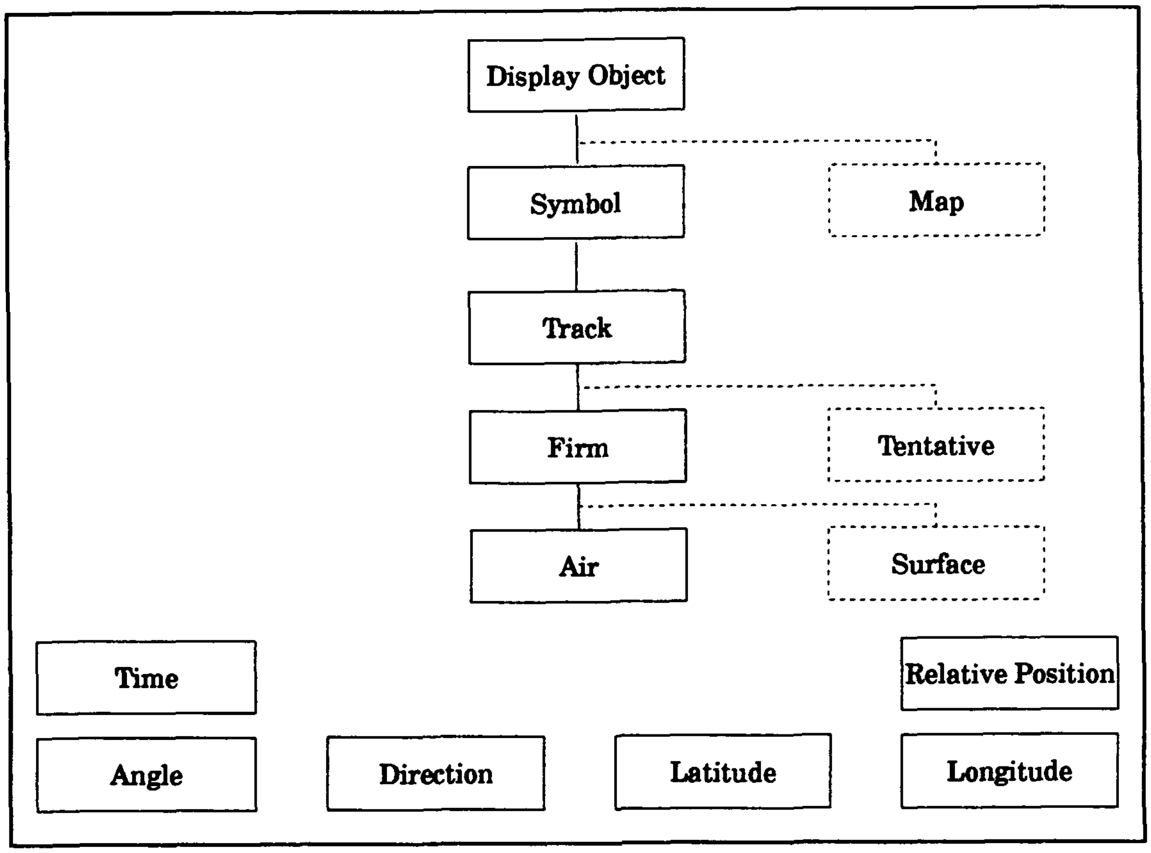


Figure 44. Object-Oriented Air Track Symbol Class Hierarchy

values. Notice that the class constants can be used like abstract data types. The methods are operations that can manipulate the object and its variables.

The main incompatibility between this object-oriented design and the rest of the software design was the Plot Symbol method listed in the SYMBOL class. The drawing function for the whole software design is located in the user interface library and not in the shared object library where the object-oriented software resides. Therefore, it is not possible to invoke a plotting method which cannot access the drawing software. The methods that were implemented are those that deal only with managing the track data and are described in Chapter IV.

7. Design Summary

This design discussion has described primarily an object-based method which uses the capabilities of two automated tools, TAE Plus and ObjectMaker. The influence of the tools

CLASS: DISPLAY OBJECT
Superclasses: none
Class Variables: none
Instance Variables:
  Display Object Number: Integer
  Position Time: Time
  Origin: (Local, Remote)
Methods:
  Create
  Delete
  Update
  CPA Processing
  Get & Set for each Variable

CLASS: SYMBOL
Superclasses: DISPLAY OBJECT
Class Variables: none
Instance Variables:
Methods:
  Plot Symbol

CLASS: TRACK
Superclasses: SYMBOL
Class Variables: none
Instance Variables:
  Position: Relative Position
  Course: Angle
  Speed: Real
  Identity: (Unknown, Friendly, Hostile
Methods:
  Get and Set for each Variable
  Calculate Position
  Course and Speed Determination

CLASS: FIRM TRACK
Superclasses: TRACK
Class Variables: none
Instance Variables: none
Methods:
  Identification Function
  Track History Processing

CLASS: AIR TRACK
Superclasses: FIRM TRACK
Class Variables: none
Instance Variables:
  Altitude: Real
Methods:
  Get and Set Altitude

Figure 45. Track Class Definitions

was significant since they allowed the automatic generation of Ada source code and design diagrams. Starting with the event-driven structure of the TAE Plus generated software, the design was modified to allow the inclusion of an external client application that could share the user interface and the data resources. Although the majority of the automatically generated design diagrams and source code templates were modified manually, the value of a tool like ObjectMaker for large projects was recognized.

The object-oriented design of the air tracks proved feasible within the shared object part of the design mainly because it was limited to the maintenance of track data in primary memory and did not involve the more complex method of plotting a track on the user interface.

The design issues that proved the most challenging involved determining how to manage the shared data and how to update the user interface. Allowing the client application access to the shared data types proved less than ideal but workable through the application library. As for whether to update the user interface manually or automatically, this responsibility was given solely to the client application.

IV. SOFTWARE IMPLEMENTATION

A. THE IMPLEMENTATION OF THE PACKAGES

The process of implementing the functions involved adding code to the software templates created during the design process. The subprograms that implemented the functions are discussed in the context of the monitor package, the application library package, the event handler library package, the user interface library package, the shared objects library package, and the track package. Each implemented routine is listed and explanations are given concerning interesting situations that surfaced during the coding. Issues that needed to be considered to implement some of the uncoded functions are described as well. Also, in order to follow along with this discussion, it might be useful to look at the design diagrams in Chapter III and the package specification listings in Appendices A and B.

1. The Monitor Package

The monitor was implemented as to support the application library functions that were implemented. The monitor package contains two main parts: a task and an event loop. The task has an entry to handle each request that the application package routines make. The other main part of the monitor is the event loop, which is inside of the task, that recognizes user-generated events and then calls the appropriate event handler. All of the user interface events are recognized, including the TAE Plus supported X workspace events. The event recognition and handler dispatching is implemented using the TAE Plus WPT library. The whole event loop was automatically created when the user interface source code was generated.

2. The Application Package

The Session Library functions implemented in the application package included Initialize_System, Display_Interface, Update_Label, and Stop_User_Interface functions. The use of each of these is shown in the example application in Appendix A.

The Initialize_System routine caused the user interface monitor to initialize the user interface and provided information that could be displayed when the interface first appeared. The alternative would have been to not provide any initial information and let the programmer call the regular routines to provide the initial values after the interfaced appeared. This approach was not used since the user would see somewhat of a piecemeal interface update when it first appeared considering that the initial map processing takes a significant amount of time.

The rest of the Session Library subprograms that were implemented had some timing issues to consider. The Display_Interface routine was needed in addition to the Initialize_System routine because task switching seemed to cause problems. What this means is the example application program had the tendency to get ahead of the monitor task before the user interface was displayed. This sometimes caused the interface to not be fully displayed while the application hogged the single processor. This was a difficult problem to track down and seemed to be solved by making sure the user interface was displayed before any other processing would be allowed.

The timing considerations of the Update_Label and Stop_User_Interface routines were not significant. The issue was just to make sure a non-drifting timer was used so the actual delay for regularly updating a label or stopping the user interface would be reasonably close to the actual delay time specified. Exact delay times could not be assured since the operating system did not support real-time processing.

The session functions that were not implemented, Get_Acknowledgments and User_Quit_Interface, would only have to check shared data on a routine basis for the appropriate information. The intention was to change the applicable shared data when the user either acknowledged an alert message or label prompt, or quit the user interface. Then the client application would just read the shared data to determine if any user action occurred. This was the approach thought best for returning data to the application from the user interface.

The symbol functions that were implemented included Add_New_Track, Delete_Track, and Get_Track_Info. The routines only updated the shared objects library data in order to test the object-oriented code. The track that is later described as being displayed on the interface was hardcoded and not displayed as a result of the Add_New_Track routine. The rest of the symbol functions were not implemented.

None of the application map functions were implemented. The issues involved to code them will not be discussed since most of the detailed design had not yet been considered.

3. The Event Handler Package

The intention was for the event handler subprograms to be organized in the session, map, and symbol categories as shown in the design diagrams. Ultimately, they were not. The only significant subprogram that was implemented was Refresh_Screen. Implementing this entailed creating a System_Call package that contained an implementation dependent routine that was used to call the external xrefresh program.

The Quit_Session function was implemented as well. The event handler routine set a flag in the shared object library which the user interface monitor checked every time while looping. When a user would select the Quit button, the quit flag was set to True and the monitor task terminated.

Some events were handled automatically by the TAE Plus interface library. One example was the help feature. It can be said that the full help system was implemented for the user interface since it was built into the TAE Plus software design and generated code. The same was true for connecting different panels to each other. The TAE Plus generated software and libraries took care of these events without any programming.

All of the other event handlers are connected to a Not_Implemented panel. Again, the issues of the non-implemented functions will not be discussed.

4. The User Interface Package

The subprograms that were written for the C2_User_Interface_Library package helped implement the application library and event handler library subprograms. As described in the design, all of the TAE Plus user interface library functions and Xlib graphics functions had to be called from subprograms in this package.

The user interface library subprograms, Set_Default_Display_ID, Set_Main_Map_Panel_Info, and Set_Max_Map_Panel_Info, were all implemented. These were called from within the interface monitor task when the application library Initialize_System routine was called by the main program. The Set_Default_Display_ID routine in-turn called the Map_Drawing package Set_ID routine to place the ID into the only package where it was used.

The implemented session library routines included Init_Label, Display_Label, and Quit_User_Interface. The label routines were called by the application library initialization and update label routines through the monitor. The quit routine was called by the quit button event handler. The quit routine was necessary to allow the event handler access to the shared object library Interface_Done flag. It was important to have the event handlers to go through the user interface package routines, instead of directly to the shared library routines, because most of the time user interface actions needed to be accomplished before or after the shared data was accessed or modified. Sometimes this seemed to create duplicate subprograms in two different packages. Although an effort was made to uniquely name the subprograms, some of them did become very similar.

The implemented map library routines included the Initialize_Map subprogram. This in-turn called the Map_Drawing routines, Init_Map, Draw_Map, and Draw_Track. The Init_Map routine also called the Read_Data procedure and Draw_Map called the Map_Utilities package's Lat_Long_To_X_Y routine. All of these subprograms were implemented in a basic form to draw an example track and pseudo map. The Booch linked-list components and list utilities were used to manage the list creation and operations [12].

5. The Shared Objects Package

The C2_Shared_Persistent_Objects_Library that was shown in the design diagrams was the latest design of what the package should look like. However, what was implemented is the previous version of the package and is reflected in the source code in Appendix B. The Access Control task was not implemented. Instead the session, map, and symbol libraries were made visible to external packages. The track objects are still internal to the symbol objects package. The data types and constants, though, were implemented as shown in the design diagram.

The routines that were implemented basically supported those that were written in the application and user interface packages. In the session package the Set and Get User_Interface_Done routines were coded. The same goes for Set and Get All_Labels.

In the map package the Set and Get Map_Info routines were implemented. The only information currently stored is the map file name. The application library initialization routine sets the map name and the user interface initialize map routine gets the map file name to open and read the file.

The routines implemented in the symbol objects package included Add_New_Track, Delete_Track, and Get_Track_Info. The only track information stored in the symbol package were track IDs since the actual objects were managed by the object-oriented code in the track package.

6. The Track Objects Package using Classic-Ada

Using Classic-Ada to implement the management of the track data worked pretty well. The code generated by the Classic-Ada preprocessor is actual Ada code. The only special part of the final code is the proprietary Ada routines that facilitate the passing of messages between objects. Appendix B lists the class specifications of the Classic-Ada code, but not the resultant pure Ada code.

The methods that were implemented included Instantiate_Track, Set_Track_Data, Get_Track_Data, and Delete_Track.

7. Creating an Example Client Application

An example application was developed that tested the implemented application library functions. This main subprogram is listed in Appendix A. Its functionality basically consists of updating the time and date in two of the labels and automatically stopping the user interface after sending a warning alert message.

The calendar utilities of the Booch components were used as an example client-provided package. The routines in this package provided the time and data information for the label displays, This package was also used to provide types for the track time type definitions.

B. SOFTWARE IMPLEMENTATION ISSUES

1. Accessing Shared Resources

The shared resource issue was again apparent during implementation. After coding had already started it was realized that some sort of concurrency control was need on the shared objects. The solution was to put an access control task in the shared object library, although it was not implemented. Another issue was that the shared data types that were placed in the shared object library forced the client application to have access to the rest of the shared library. This was not ideal since the application could directly call the shared object library routines. However, no suitable alternative was found to get around this situation.

2. Coding and Documentation Styles

Since the generated source code from TAE Plus and ObjectMaker did not follow the same coding standards, an effort was made to follow the guidelines offered by the Software Productivity Consortium (SPC) [1]. To its credit the TAE Plus did follow a coding style that was called the "1987 NASA guidelines." The ObjectMaker documentation did not claim the product's generated code followed any kind of standard. The style was, however, consistent and well structured. The documentation style for the thesis software also came from the SPC guidelines.

C. IMPLEMENTATION SUMMARY

In general the implementation just got underway. However, this implementation should provide a good starting point for further work in this area

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

Looking back on the research and development of this thesis, there were several things learned in the areas of the software development process and software reuse that should be useful for future work.

1. The Software Development Process

Specifying the requirements and designing the user interfaces was a very time-intensive process. There seems to be a strong argument for having a person dedicated to user interface issues on a software development team. In fact Tiburon Systems, Inc., is one company that has a full-time user-interface designer [60].

a. Requirements

For the software requirements, additional formal specifications for the user interface interaction procedures might have proved useful for the storyboard interaction descriptions. The feature chart proved useful for depicting the capabilities and general interaction of the user interface. A formal event-response description to accompany the feature chart and storyboards would have been a good addition.

b. Design

The automated tools could have been used more in the design process. As mentioned, some source code templates were created before reverse-engineering the diagrams for the user interface code. With experience in using the tools, very little manual coding should be needed to create a general template of almost any Ada software design. There is now a newer version of ObjectMaker available that should make it easier to use for making changes and for using other diagram techniques. Unfortunately this newer version did not arrive in time to review its capabilities or be used for this thesis. Also, a newer version of TAE Plus should be released later this year that implements more of the OSF/Motif widgets and other features such as scrollable X workspaces. With these capabilities, certain graphics functions do not need to be designed in as much detail since they would already be part of the automatically generated user interface.

c. *Implementation*

As described in Chapter IV, the implementation of the map, symbol, and session functions included a subset of the functions listed in the requirements so there is plenty of work that can still be done. The Air Force Rome Laboratory now has a comprehensive set of map manipulation functions written in Ada that could probably be used in a follow-on project. See Appendix C for a point of contact for this software.

2. The Software Reuse Issues

For the software in this thesis, an effort was made to find and reuse software that could help in both the design and implementation. From previous thesis work and the sources listed in Appendix C, software was found and informally evaluated for use in this work. Some problems were encountered along with some useful software.

One problem encountered was when trying to reuse software in this thesis from previous theses that used a similar software development tool (e.g. TAE Plus). The problem was that the tool had changed enough from the previous version that converting the old software to work with the upgraded tool did not seem worthwhile. The LCCDS thesis [56] had used the previous version of TAE Plus and to modify the user interface code developed with the older version for this thesis did not seem advisable. One reason was that most of the interface panels and items required extensive use of Xlib. Perhaps upgrading the user interface of this thesis to the next version of TAE Plus will prove just as difficult—hopefully not.

Even though the LCCDS user interface was not very reusable, some of the interface software proved useful because it provided excellent examples of how to do X graphics programming using Ada. In addition, several algorithms for some of the functions mentioned in this thesis are coded in the LCCDS thesis.

Other software that proved useful included some unpublished software that manipulated and drew maps using X, but were written in the C language [62]. Some of the software, available from the TAE support office (see Appendix C) also had a TAE Plus user interface. Although this C software provided good examples of working with map data and using X, it was generally difficult to read and follow since it was not intended to be reused by anyone else but the original author.

The other sources of Ada software listed in Appendix C may have more source code that would be useful for a follow-on project to this thesis. However, they were not fully investigated, mainly because of the time-consuming process of trying to find and retrieve useful software. The growing number of Ada software repository projects should make this process more effective for future work.

B. RECOMMENDATIONS FOR FURTHER STUDY

Future work in the area of command and control Ada software for map manipulation could focus on expanding the common function list, using better development tools to make implementation of the functions easier, and taking advantage of other advanced research.

1. Expand the Common Function List

It would be useful to expand the common function list to include functions that support map and symbol manipulation. Although some of these complementary functions were identified in the related C^2 and window functions, they were not a focus of the this thesis. Some of these functions, such as message handling, were only provided through an external interface. Providing capabilities for message handling and persistent data storage as integral parts of the user interface would make it more useful and easier to use. Some of these capabilities are described and partially implemented in a couple of sources [9, 31, 56].

2. Implement Functions with Better Tools

As previously mentioned, both TAE Plus and ObjectMaker are being upgraded. These kinds of tools make design and implementation of software progress much more easily. Obviously other similar tools should be sought to take advantage of new capabilities. Better tools are bound to exist. Being able to get them to use is obviously another matter.

Other tools to consider are those that are tailored to provide a specific function such as on-line help. Although TAE Plus provides an integrated help feature, it is not well suited for providing a users' manual where large amounts of information need to be managed. A hypertext system such as the one used in the FrameMaker [24] desktop publishing system might be a good candidate.

At a lower level, some of the newer X-based toolkit libraries could be used to implement functions that would be too difficult in just Xlib. The software layer foundation that this thesis software was built upon was not fully exploited. TAE Plus did use some of these toolkit libraries, but no effort was made to use them to draw maps or symbols in the X workspace of the user interface.

3. Other Research

a. Three-Dimensional Graphics

As discussed throughout the workstation of the future report, 3D graphics are at the leading edge for C^2 workstations [38]. Using the PHIGS extension to X (PEX) to add portable 3D graphics to this C^2 interface would something to consider in a follow-on project. Although the current version of X (X11R5) includes PEX, the next version of X (X11R6, due out in mid-1992) will supposedly be easier to use. The current complaint is that application programming interface for PEX is tedious and needs work. Also, some PEX toolkits efforts are in progress which should make programming 3D graphics in the X environment much easier to do in the near future.

b. The Air Force Rome Laboratory

The Air Force Rome Laboratory is doing a lot of work in C^2 map systems. Although several documents were used from and personal contact was made with the lab, this resource could be used much more in future work.

C. LAST THOUGHTS

The pursuit of developing a fully functional C^2 map user interface written in Ada is still a worthwhile goal. Researchers and students would benefit by being able to easily develop a map-based interface to help visualize new situation-monitoring and enhanced decision-making techniques that will help the joint commanders of the future.

APPENDIX A

APPLICATION PROGRAMMER'S GUIDE

This guide lists the Ada specifications for the C2 Application Library and the C2 Shared Persistent Objects Library types to show an application programmer what functions are available to integrate a client application with the user interface. Following the specificiations is an example client application, called Run_C2_Map, that shows how to call the library functions that were actually implemented.

The key item to note in the example application is the intialization steps. The Initialize_System and Display_Interface procedures must be called before any other application library procedures can be called. After that, any of the library procedures can be called. As mentioned in Chapter IV, only certain functions were actually implemented and all of them are shown in the example program. The rest of the procedures were intended to work in a similar way.

The data types that are shown in the application library procedure parameter lists can be found in the C2_Shared_Lib package that follows the C2_Appl_Lib.

C2 Application Library

```
-- File:    C2_Appl_s.a
-- Author:  Bennett K. Larson (bkl)
-- System:  SparcStation 2, SunOS 4.1.2
-- Compiler: Verdix Ada Development System (VADS) 6.0
-- Revision History:
-- 15 Jun 92   bkl
--   - Final Version
-- 29 Mar 92   bkl
--   - Original version

with Calendar_Utilities;
with C2_Shared_Lib;
---------------------------------------------------------------------
-- C2_Appl_Lib
---------------------------------------------------------------------
-- Purpose:
--   This package encompasses the three user application callable library
--   packages, Session, Map, and Symbol.
--
-- Effects:
--   - The expected usage is:
--     1. The user Application is to "With in" this package to have visibility
--        to all needed functions associated with the C2 user interface.
---------------------------------------------------------------------
package C2_Appl_Lib is


 ---------------------------------------------------------------------
 -- These packages are made visible to allow access to common data types
 -- used both by the application and user interface libraries.
 ---------------------------------------------------------------------
 package Shared renames C2_Shared_Lib;

 Max_Entry_Select_Delay : Duration := 0.1; -- Default for all entry calls
                                           -- to the Monitor Task. Can be
                                           -- set by user application using
                                           -- Initialize_System call.

 ---------------------------------------------------------------------
 package Session_Appl_Lib is

  use Shared;
```

```
------------------------------------------------------------------
-- Initialize_System
------------------------------------------------------------------
-- Purpose:
--  This procedure causes the user interface panels to be initialized
--  and gives the event loop timing values.
-- Exceptions:
--  None.
------------------------------------------------------------------
procedure Initialize_System
        ( Entry_Select_Delay   : in Duration;
          Event_Select_Delay   : in Duration;
          Event_Process_Time   : in Duration;
          Label_Init_List      : in Label_List_Type;
          Map_Info             : in Map_Info_Type );


------------------------------------------------------------------
-- Display_Interface
------------------------------------------------------------------
-- Purpose:
--  This procedure displays the user interface after any initial values
--  are set by the user application. This procedure ensures the
--  application program gives up the processor so the user interface
--  has time to be properly initialized by the call to Initialize_System.
-- Exceptions:
--  None.
------------------------------------------------------------------
procedure Display_Interface;


------------------------------------------------------------------
-- Update_Label
------------------------------------------------------------------
-- Purpose:
--  This procedure updates the information of the specified label on the
--  user interface.
-- Exceptions:
--  TASKING_ERROR - if an entry call is pending and the user
--    interface is terminated, the resulting error is captured and
--    ignored.
------------------------------------------------------------------
procedure Update_Label
        ( Label_Contents : in Label_Contents_Type;
          Label_Position : in Label_Position_Type );
```

```
-----------------------------------------------------------------
-- Send_Alert_Message
-----------------------------------------------------------------
-- Purpose:
--   This procedure displays an alert message in one of the alert areas.
-- Exceptions:
--   None.
-----------------------------------------------------------------
procedure Send_Alert_Message
        ( Message : in Alert_Message_Type );


-----------------------------------------------------------------
-- Get_Alert_Acknowledgement
-----------------------------------------------------------------
-- Purpose:
--   This procedure checks to see if the user has acknowledged an alert
--   message that was set to be acknowledged.
-- Exceptions:
--   None.
-----------------------------------------------------------------
function Get_Alert_Acknowledgement
        return Alert_Acknowledgement_Type;


-----------------------------------------------------------------
-- User_Quit_Interface
-----------------------------------------------------------------
-- Purpose:
--   This function allows the application to know when the user has
--   terminiated the map interface.
-- Exceptions:
--   None.
-----------------------------------------------------------------
function User_Quit_Interface
        return Boolean;
```

```
  ----------------------------------------------------------------
  -- Stop_User_Interface
  ----------------------------------------------------------------
  -- Purpose:
  --  This procedure stops the user interface after sending a Critical Alert
  --  Message and delaying a period of time specified by the programmer.
  -- Exceptions:
  --  None.
  ----------------------------------------------------------------
  procedure Stop_User_Interface
        ( Shutdown_Message : in Alert_Message_Type;
          Delay_Time       : in Duration );


end Session_Appl_Lib;


-----------------------------------------------------------------
package Map_Appl_Lib is

  use Shared;

  ----------------------------------------------------------------
  -- Set_New_Map_Info
  ----------------------------------------------------------------
  -- Purpose:
  --  This procedure specifies a new map to display.
  -- Exceptions:
  --  None.
  ----------------------------------------------------------------
  procedure Set_New_Map_Info
        ( Map_Info : in Map_Info_Type );

  ----------------------------------------------------------------
  -- Get_Map_Info
  ----------------------------------------------------------------
  -- Purpose:
  --  This procedure returns information about the map currently being
  --  displayed.
  -- Exceptions:
  --  None.
  ----------------------------------------------------------------
  function Get_Map_Info
        return Map_Info_Type;

end Map_Appl_Lib;
```

```
---------------------------------------------------------------------
package Symbol_Appl_Lib is

 use Shared;

 --------------------------------------------------------------------
 -- Add_New_Track
 --------------------------------------------------------------------
 -- Purpose:
 --   This procedure adds a new track to the list of tracks.  It will be
 --   displayed if it is within bounds of the current map and if the
 --   classification is appropriate.
 -- Exceptions:
 --   None.
 --------------------------------------------------------------------
 procedure Add_New_Track
          ( Track_Info : in Track_Info_Type );


 --------------------------------------------------------------------
 -- Delete_Track
 --------------------------------------------------------------------
 -- Purpose:
 --   This procedure deletes a track from the track list.
 -- Exceptions:
 --   None.
 --------------------------------------------------------------------
 procedure Delete_Track
          ( Track_ID : in Track_ID_Type );


 --------------------------------------------------------------------
 -- Get_Track_Info
 --------------------------------------------------------------------
 -- Purpose:
 --   This procedure gets information about a certain track.
 -- Exceptions:
 --   None.
 --------------------------------------------------------------------
 function Get_Track_Info
          ( Track_ID   : in  Track_ID_Type) return Track_Info_Type;
```

```
   -----------------------------------------------------------------
   -- Update_Track_Info
   -----------------------------------------------------------------
   -- Purpose:
   --   This procedure updates the information about a track.
   -- Exceptions:
   --   None.
   -----------------------------------------------------------------
   procedure Update_Track_Info
          ( Track_Info : in Track_Info_Type );

   -----------------------------------------------------------------
   -- Confirm_Manual_Track_Addition
   -----------------------------------------------------------------
   -- Purpose:
   --   This procedure allows the user application to control whether or
   --   not a track can be added to the track list from the user interface.
   -- Exceptions:
   --   None.
   -----------------------------------------------------------------
   procedure Confirm_Manual_Track_Addition
          ( Track_ID : out Track_ID_Type );


   -----------------------------------------------------------------
   -- Confirm_Manual_Track_Deletion
   -----------------------------------------------------------------
   -- Purpose:
   --   This procedure allows the user application to control whether or
   --   not a track should be deleted from the track list.
   -- Exceptions:
   --   None.
   -----------------------------------------------------------------
   procedure Confirm_Manual_Track_Deletion
          ( Track_ID : out Track_ID_Type );

  end Symbol_Appl_Lib;


end C2_Appl_Lib;
```

C2 Shared Persistent Objects Library

(Types Only)

```
-- File:    C2_Shared_Lib_s.a
-- Author:  Bennett K. Larson (bkl)
-- System:  SparcStation 2, SunOS 4.1.2
-- Compiler: Verdix Ada Development System (VADS) 6.0
-- Revision History:
-- 11 Jun 92   bkl
--   - Final version
-- 25 May 92   bkl
--   - Original version


With Calendar_Utilities;
---------------------------------------------------------------------
-- C2_Shared_Lib
---------------------------------------------------------------------
-- Purpose:
--   This package contains shared types and objects for the application and UI
--   packages, Session, Map, and Symbol.
---------------------------------------------------------------------
package C2_Shared_Lib is


   ------------------------------------------------------------------
   -- Label Types
   ------------------------------------------------------------------
   Max_Label_Contents_Length : constant Natural := 28;
   Max_Label_Positions       : constant Natural := 5;

   subtype Label_Contents_Type is String (1..Max_Label_Contents_Length);
   subtype Label_Position_Type is Natural range 1.. Max_Label_Positions;

   type Label_List_Type is array
        (1..Max_Label_Positions) of Label_Contents_Type;

   ------------------------------------------------------------------
   -- Alert Message Types
   ------------------------------------------------------------------
   Max_Alert_Number_Each_Category : constant Natural := 999;
   Alert_Text_Len                 : constant Integer := 20;
   type Alert_Category_Type is (CRITICAL, CAUTION, INFO);
type Alert_Message_Type is
     record
       Number : Natural range 0..Max_Alert_Number_Each_Category;
       Text   : String (1..Alert_Text_Len)
                  := (1..Alert_Text_Len => ' ');
       Alert_Category : Alert_Category_Type;
       Acknowledgement_Required : Boolean;
     end record;

type Alert_Acknowledgement_Type is
     record
       Number         : Natural range 0..Max_Alert_Number_Each_Category;
       Alert_Category : Alert_Category_Type;
     end record;

Max_File_Name_Len : constant Integer := 256;

subtype Map_Name_Type is String (1..Max_File_Name_Len);

type Map_Info_Type is
     record
       Map_File_Name : Map_Name_Type := (1..Max_File_Name_Len => ' ');
       Map_File_Name_Len : Natural;

     end record;


  ---------------------------------------------------------------------
-- Track Types
---------------------------------------------------------------------
Max_Track_ID : constant Positive := 100;

subtype Track_ID_Type is Positive range 1..Max_Track_ID;

subtype Degree_Type is Natural range 0..359;
subtype Minute_Type is Natural range 0..59;
subtype Second_Type is Natural range 0..59;

subtype Range_Type  is Float range 0.0..Float'LAST;
subtype Speed_Type  is Float range 0.0..Float'LAST;
subtype Height_Type is Float;

subtype Zone_Type is Character range 'A'..'Z';

type Angle_Type is
     record
       Degree : Degree_Type;
       Minute : Minute_Type;
       Second : Second_Type;
     end record;
```

```
type Latitude_Hemisphere_Type  is (NORTH, SOUTH);
type Longitude_Hemisphere_Type is (EAST, WEST);

type Latitude_Type is
    record
     Angle      : Angle_Type;
     Hemisphere : Latitude_Hemisphere_Type;
    end record;

type Longitude_Type is
    record
     Angle      : Angle_Type;
     Hemisphere : Longitude_Hemisphere_Type;
    end record;

type Position_Type is
    record
     Latitude  : Latitude_Type;
     Longitude : Longitude_Type;
    end record;

type Position_Time_Type is
    record
     The_Time : Calendar_Utilities.Time;
     Zone     : Zone_Type;
    end record;


type North_Type is (TRUE, MAGNETIC);

type Bearing_Type is
    record
     Direction_Angle : Angle_Type;
     Reference_North : North_Type;
    end record;


type Relative_Position_Type is
    record
     Bearing   : Bearing_Type;
     The_Range : Range_Type;
    end record;

type Origin_Type is (REMOTE, LOCAL_AUTO, LOCAL_MANUAL);
type Identity_Type is (UNKNOWN, FRIENDLY, HOSTILE);
```

```
    type Track_Info_Type is
        record
          Track_ID          : Track_ID_Type;
          Position          : Position_Type;
          Position_Time     : Position_Time_Type;
          Relative_Position : Relative_Position_Type;
          Origin            : Origin_Type;
          Course            : Angle_Type;
          Speed             : Speed_Type;
          Identity          : Identity_Type;
          Height            : Height_Type;
        end record;

end C2_Shared_Lib;
```

Example Client Application

```
-- File:    Run_C2_Map.a   (Test Program)
-- Author:  Bennett K. Larson (bkl)
-- System:  SparcStation 2, SunOS 4.1.2
-- Compiler: Verdix Ada Development System (VADS) 6.0
-- Revision History:
-- 15 Jun 92   bkl
--   - Final version
-- 25 May 92   bkl
--   - added documentation and stop message example
--    Mar 92   bkl
--   - Original version


with Calendar; use Calendar;     -- "use" for operator (e.g. +) visibility
with Calendar_Utilities;         -- User provided (Booch components)
with C2_Appl_Lib;
with Text_IO;
use  Text_IO;

-----------------------------------------------------------------------
-- Run_C2_Map
-----------------------------------------------------------------------
-- Purpose:
--   This procedure is the main subprogram for the test user application.
-- Implementation Notes: The time, date, and map are intialized and then the
--   time is updated every minute.  A non-drifting time delay is used.
--   All values of type Duration are in units of seconds.
--
-- Exceptions:
--   None.
-----------------------------------------------------------------------
procedure Run_C2_Map is

  package Int_IO is new Integer_IO (Integer);
  use Int_IO;

  package Flt_IO is new Float_IO (Float);
  use Flt_IO;

  package C2 renames C2_Appl_Lib;

  Military_Time_String_Length : constant Integer := 11; -- HH:MM:SS:ss
  Short_Time_String_Length    : constant Integer := 5;  -- HH:MM
  MDY_Date_String_Length      : constant Integer := 8;  -- MM/DD/YY

  Map_Name_String  : constant String :=  "map/us.amd";
```

```
Initial_Map_Name : constant C2.Shared.Map_Name_Type :=
      Map_Name_String &
    ( Map_Name_String'LENGTH + 1..C2.Shared.Max_File_Name_Len => ' ');


Initial_Map_Info : C2.Shared.Map_Info_Type;

Label_Init_List      : C2.Shared.Label_List_Type;
Blank_Label_Contents : String
                (1..C2.Shared.Max_Label_Contents_Length) :=
                (1..C2.Shared.Max_Label_Contents_Length => ' ');

First_Time : Calendar.Time;
Next_Time  : Calendar.Time;

Future_Time       : Calendar.Time;
Clock_Update_Wait : Duration := 15.0;

Appl_Delay        : Duration;
Interval          : constant Duration := 1.0;

----------------------------------------------------------------------
-- Delay durations needed to limit time for user interface processing and
-- making entry calls so user application can handle critical processing
-- on a scheduled basis.
--
-- Note:  This are arbitrary defaults.
----------------------------------------------------------------------
Entry_Select_Delay : Duration := 0.10;
Event_Select_Delay : Duration := 0.10;
Event_Process_Time : Duration := 1.0;

----------------------------------------------------------------------
-- There are five application-defineable labels.
----------------------------------------------------------------------
Time_Label   : C2.Shared.Label_Position_Type := 1;
Date_Label   : C2.Shared.Label_Position_Type := 2;
Blank_LabelA : C2.Shared.Label_Position_Type := 3;
Blank_LabelB : C2.Shared.Label_Position_Type := 4;
Blank_LabelC : C2.Shared.Label_Position_Type := 5;

Current_Time        : Calendar_Utilities.Time;
Current_Time_String : C2.Shared.Label_Contents_Type;
```

```
-------------------------------------------------------------------
-- Stop User Interface Info
-------------------------------------------------------------------
Stop_UI_Countup   : Integer := 0;
Stop_Message      : C2.Shared.Alert_Message_Type :=
                    ( Number => 0,
                      Text   => "SHUTDOWN IN 15 SECS ",
                      Alert_Category  => C2.Shared.CRITICAL,
                      Acknowledgement_Required => FALSE );
Stop_Delay_Time   : Duration := 15.0;

-------------------------------------------------------------------
-- Add New Track Info
-------------------------------------------------------------------
New_Track_Info : C2.Shared.Track_Info_Type;

-------------------------------------------------------------------
-- Get Track Info
-------------------------------------------------------------------
Get_Track_ID   : C2.Shared.Track_ID_Type;
Get_Track_Info : C2.Shared.Track_Info_Type;

------------------------------------------------------------------
-- Get_Time_String
------------------------------------------------------------------
-- Purpose:
--   This function returns the current time in a form for displaying in
--   a user label of the interface.
-- Implementation Notes:
--
-- Exceptions:
--   None.
------------------------------------------------------------------
function Get_Time_String
         return C2.Shared.Label_Contents_Type is

  Short_Time_String : C2.Shared.Label_Contents_Type
              := (1..C2.Shared.Max_Label_Contents_Length=> ' ');

  Full_Time_String  : C2.Shared.Label_Contents_Type;

  Temp_Time_String  : String (1..C2.Shared.Max_Label_Contents_Length);

  Time_Image  : String (1..Military_Time_String_Length);
  Time_Clock  : Calendar_Utilities.Time;

begin -- Get_Time_String

  Time_Clock  := Calendar_Utilities.Time_Of(Calendar.Clock);
  Time_Image  := Calendar_Utilities.Time_Image_Of(Time_Clock,
            Calendar_Utilities.Military);

  -------------------------------------------------------------------
  -- Two step type conversion is necessary
  -------------------------------------------------------------------
  Temp_Time_String (1..Military_Time_String_Length) := Time_Image;

  Full_Time_String := C2.Shared.Label_Contents_Type(Temp_Time_String);

  Short_Time_String (1..Short_Time_String_Length) :=
                   Full_Time_String (1..Short_Time_String_Length);
  return Short_Time_String;

end Get_Time_String;

-------------------------------------------------------------------
-- Get_Time
-------------------------------------------------------------------
-- Purpose:
--   This procedure intializes a current time variable and also puts it
--   into a string for later use.
-- Implementation Notes:
--
-- Exceptions:
--   None.
-------------------------------------------------------------------
Procedure Get_Time
         (The_New_Time : out Calendar_Utilities.Time;
          Time_String  : out C2.Shared.Label_Contents_Type) is

  Short_Time_String : C2.Shared.Label_Contents_Type
               := (1..C2.Shared.Max_Label_Contents_Length=> ' ');

  Full_Time_String  : C2.Shared.Label_Contents_Type;

  Temp_Time_String  : String (1..C2.Shared.Max_Label_Contents_Length);

  Time_Image  : String (1..Military_Time_String_Length);
  Time_Clock  : Calendar_Utilities.Time;

begin  -- Get_Time

  Time_Clock  := Calendar_Utilities.Time_Of(Calendar.Clock);
  Time_Image  := Calendar_Utilities.Time_Image_Of(Time_Clock,
             Calendar_Utilities.Military);

  Temp_Time_String (1..Military_Time_String_Length) := Time_Image;

  Full_Time_String := C2.Shared.Label_Contents_Type(Temp_Time_String);

  Short_Time_String (1..Short_Time_String_Length) :=
                  Full_Time_String (1..Short_Time_String_Length);

  The_New_Time := Time_Clock;
  Time_String  := Short_Time_String;

end Get_Time;

--------------------------------------------------------------------
-- Get_Date_String
--------------------------------------------------------------------
-- Purpose:
--   This function returns the current date in a string ready for display
--   in a user interface label.
-- Exceptions:
--   None.
--------------------------------------------------------------------
function Get_Date_String
        return C2.Shared.Label_Contents_Type is

  Date_String : C2.Shared.Label_Contents_Type
          := (1..C2.Shared.Max_Label_Contents_Length => ' ');

  Date_Image  : String (1..MDY_Date_String_Length);
  Time_Clock  : Calendar_Utilities.Time;

  Temp_Date_String  : String (1..C2.Shared.Max_Label_Contents_Length)
              := (1..C2.Shared.Max_Label_Contents_Length=> ' ');

begin  -- Get_Date_String

  Time_Clock  := Calendar_Utilities.Time_Of(Calendar.Clock);
  Date_Image  := Calendar_Utilities.Date_Image_Of(Time_Clock,
            Calendar_Utilities.Month_Day_Year);

  Temp_Date_String (1..MDY_Date_String_Length) := Date_Image;

  Date_String := C2.Shared.Label_Contents_Type(Temp_Date_String);

  return Date_String;

end Get_Date_String;
```

```
--------------------------------------------------------------------
-- Update_Time_Date
--------------------------------------------------------------------
-- Purpose:
--   This procedure updates the time every minute and checks to see if
--   the date also needs changing.
-- Implementation Notes:
--   The whole time and date string are replaced. An enhancement
--   would be to only update digits that changed. However, the TAE label type
--   would have to be changed to make this work. Updating a label has
--   to be done all at once. Also, by changing the label type,
--   flexibility would be lost to use the label for other types of
--   information.
--
-- Exceptions:
--   None.
--------------------------------------------------------------------
procedure Update_Time_Date is

  Time_String : C2.Shared.Label_Contents_Type;
  New_Time : Calendar_Utilities.Time;

begin -- Update_Time_Date

  Get_Time (New_Time, Time_String);

  if Integer (Current_Time.The_Minute) /= Integer (New_Time.The_Minute) then
    C2.Session_Appl_Lib.Update_Label (Time_String, Time_Label);
  end if;

  if Integer (Current_Time.The_Day) /= Integer (New_Time.The_Day) then
    C2.Session_Appl_Lib.Update_Label (Get_Date_String, Date_Label);
  end if;

  Current_Time := New_Time;
  Current_Time_String := Time_String;

end Update_Time_Date;
```

```
------------------------------------------------------------------------
begin  -- Run_C2_Map

  ----------------------------------------------------------------------
  -- Initializes Current_Time so should be the first procedure called.
  ----------------------------------------------------------------------
  Get_Time (Current_Time, Current_Time_String);

  Label_Init_List (Time_Label) := Get_Time_String;
  Label_Init_List (Date_Label) := Get_Date_String;
  Label_Init_List (Blank_LabelA) := Blank_Label_Contents;
  Label_Init_List (Blank_LabelB) := Blank_Label_Contents;
  Label_Init_List (Blank_LabelC) := Blank_Label_Contents;

  Initial_Map_Info.Map_File_Name     := Initial_Map_Name;
  Initial_Map_Info.Map_File_Name_Len := Map_Name_String'LENGTH;

  C2.Session_Appl_Lib.Initialize_System
    ( Entry_Select_Delay,
      Event_Select_Delay,
      Event_Process_Time,
      Label_Init_List,
      Initial_Map_Info );


  C2.Session_Appl_Lib.Display_Interface;  -- create the initial panels

  First_Time := Interval + Calendar.Clock;
  Next_Time  := First_Time;


  ----------------------------------------------------------------------
  -- Add Track 1
  ----------------------------------------------------------------------
  New_Track_Info.Track_ID := 1;
  New_Track_Info.Speed    := 500.0;
  C2.Symbol_Appl_Lib.Add_New_Track
                ( New_Track_Info );

  ----------------------------------------------------------------------
  -- Add Track 2
  ----------------------------------------------------------------------
  New_Track_Info.Track_ID := 2;
  New_Track_Info.Speed    := 600.0;
  C2.Symbol_Appl_Lib.Add_New_Track
                ( New_Track_Info );
```

```
-----------------------------------------------------------------------
-- Get Track 1 info
-----------------------------------------------------------------------
Get_Track_ID := 1;
Get_Track_Info := C2.Symbol_Appl_Lib.Get_Track_Info
                        ( Get_Track_ID );

New_Line;
Put ("TEST PROG: Got Track 1 Info -- Returned Track Object Number is : ");
Put ( Get_Track_Info.Track_ID, 4 );
New_Line;

Put ("TEST PROG: Returned Track 1 Speed is: ");
Put ( Get_Track_Info.Speed, 6,2,0 );
New_Line;


-----------------------------------------------------------------------
-- Get Track 2 info
-----------------------------------------------------------------------
Get_Track_ID := 2;
Get_Track_Info := C2.Symbol_Appl_Lib.Get_Track_Info
                        ( Get_Track_ID );

New_Line;
Put ("TEST PROG: Got Track 2 Info -- Returned Track Object Number is : ");
Put ( Get_Track_Info.Track_ID, 4 );
New_Line;

Put ("TEST PROG: Returned Track 2 Speed is: ");
Put ( Get_Track_Info.Speed, 6,2,0 );
New_Line;

-----------------------------------------------------------------------
-- Update Track 1 info
-----------------------------------------------------------------------
Get_Track_ID := 1;
Get_Track_Info.Speed    := 100.0;
C2.Symbol_Appl_Lib.Update_Track_Info
            ( Get_Track_Info );

New_Line;
Put_Line ("TEST PROG: Updated Track 1 Info with new speed ");
```

```
----------------------------------------------------------------------
-- Update Track 2 info
----------------------------------------------------------------------
Get_Track_ID := 2;
Get_Track_Info.Speed    := 200.0;
C2.Symbol_Appl_Lib.Update_Track_Info
            ( Get_Track_Info );

New_Line;
Put_Line ("TEST PROG: Updated Track 2 Info with new speed ");


----------------------------------------------------------------------
-- Get Track 1 info again
----------------------------------------------------------------------
Get_Track_ID := 1;
Get_Track_Info := C2.Symbol_Appl_Lib.Get_Track_Info
                        ( Get_Track_ID );

New_Line;
Put ("TEST PROG: Got Track 1 Info AGAIN, ID is : ");
Put ( Get_Track_Info.Track_ID, 4 );
New_Line;

Put ("TEST PROG: Returned Track 1 Speed is: ");
Put ( Get_Track_Info.Speed, 6,2,0 );
New_Line;



----------------------------------------------------------------------
-- Get Track 2 info again
----------------------------------------------------------------------
Get_Track_ID := 2;
Get_Track_Info := C2.Symbol_Appl_Lib.Get_Track_Info
                        ( Get_Track_ID );

New_Line;
Put ("TEST PROG: Got Track 2 Info AGAIN, ID is : ");
Put ( Get_Track_Info.Track_ID, 4 );
New_Line;

Put ("TEST PROG: Returned Track 2 Speed is: ");
Put ( Get_Track_Info.Speed, 6,2,0 );
New_Line;
```

```
 ------------------------------------------------------------------
 -- Delete Track 1
 ------------------------------------------------------------------
 Get_Track_ID := 1;
 C2.Symbol_Appl_Lib.Delete_Track
             ( Get_Track_ID);
 New_Line;
 Put_Line ("TEST PROG: Deleted Track 1 ");


 ------------------------------------------------------------------
 -- Delete Track 2
 ------------------------------------------------------------------
 Get_Track_ID := 2;
 C2.Symbol_Appl_Lib.Delete_Track
             ( Get_Track_ID);
 New_Line;
 Put_Line ("TEST PROG: Deleted Track 2 ");


SYSTEM_LOOP:

 loop

 ------------------------------------------------------------------
 -- Delay with no drift.
 ------------------------------------------------------------------
  Appl_Delay := Next_Time - Calendar.Clock;
  delay Appl_Delay;

  Update_Time_Date;

  if C2.Session_Appl_Lib.User_Quit_Interface then

    exit;

  else

    Next_Time := Next_Time + Interval;

  end if;
```

```
    ------------------------------------------------------------------
    -- Send a warning message after 15 secs
    ------------------------------------------------------------------
    if Stop_UI_Countup > 15 then
    C2.Session_Appl_Lib.Stop_User_Interface ( Stop_Message, Stop_Delay_Time );
     null;
    end if;

    Stop_UI_Countup := Stop_UI_Countup + 1;

  end loop SYSTEM_LOOP;


end Run_C2_Map;
```

APPENDIX B

SOFTWARE SPECIFICATIONS

This appendix lists the specifications of the rest of the packages not directly related to the client application. These packages include the C2 Monitor, the C2 Event Handler Library, the C2 User Interface Library, the C2 Shared Persistent Objects Library, and the object-oriented Classic-Ada class specifications.

C2 Monitor

```
-- File:    C2_Monitor_s.a
-- Author:  Bennett K. Larson (bkl)
-- System:  SparcStation 2, SunOS 4.1.2
-- Compiler: Verdix Ada Development System (VADS) 6.0
-- Revision History:
-- 11 Jun 92   bkl
--   - Final version
-- 24 May 92   bkl
--   - Changed file name from C2_Map_s.a to C2_Monitor_s.a
--   - Added comments and other name changes
-- 29 Mar 92   bkl
--   - Original version


---------------------------------------------------------------------
-- C2_Monitor
---------------------------------------------------------------------
-- Purpose:
--   This package specifies the task used for access to the user interface.
--   Since the user interface makes many non-reentrant X Window System calls,
--   tight control of exteral access must be maintained.
--
-- Effects:
--   - The expected usage is:
--     1. Call only from the library packages (i.e. hide from client
--        applications).
--
-- Performance:
--   - Some timing delays are provided to specify  maximum task execution time.
--     See the task entries for details.
---------------------------------------------------------------------
package C2_Monitor is

  -------------------------------------------------------------------
  -- Event_Loop
  -------------------------------------------------------------------
  -- Purpose:
  --   The loop task monitors the user interface events and client
  --    application calls.
  -- Exceptions:
  --   None.
  -------------------------------------------------------------------
  task Event_Loop is

    ---------------------------------------------------------------
    -- This entry allows the maximum wait and processing times (approx.) to be
    -- specified for the select and event loop statements.
    ---------------------------------------------------------------
```

```
entry Initialize_System
      ( Event_Select_Delay : in Duration;
       Event_Process_Time : in Duration );

-----------------------------------------------------------------
-- This entry is called after the initial main map panel values have
-- been set by the client application.  This allows the user interface
-- to start with an initial map and other values (such as  label
-- definitions).
-----------------------------------------------------------------
entry Display_Interface;

-----------------------------------------------------------------
-- This entry is for label updates.
-----------------------------------------------------------------
entry Update_Label
      ( Label_Position : in Integer );

-----------------------------------------------------------------
-- This entry is to cause the alerts in a particular category to be
-- displayed.
-----------------------------------------------------------------
entry Display_Alert_Message
      ( Alert_Category : Integer );

-----------------------------------------------------------------
-- This entry is for changing the map being displayed.
-----------------------------------------------------------------
entry Change_Map;

-----------------------------------------------------------------
-- This entry is for displaying a new track
-----------------------------------------------------------------
entry Display_Track
      (Track_ID : in integer );

-----------------------------------------------------------------
-- This entry is for deleting a track from the display.
-----------------------------------------------------------------
entry Delete_Track
      (Track_ID : in integer );

-----------------------------------------------------------------
-- This entry is for updating a track position on the display.
-----------------------------------------------------------------
entry Update_Track
      (Track_ID : in integer );
```

```
    ------------------------------------------------------------------
    -- This entry allows the application to stop the user interface.
    ------------------------------------------------------------------
    entry Stop_The_Interface;

  end Event_Loop;

end C2_Monitor;
```

C2 Event Handler Library

```
-- File:    C2_Event_s.a
-- Author:  Bennett K. Larson (bkl)
-- System:  SparcStation 2, SunOS 4.1.2
-- Compiler: Verdix Ada Development System (VADS) 6.0
-- Revision History:
-- 11 Jun 92   bkl
--   - Final Version
-- 25 May 92   bkl
--   - restructured
-- 12 Apr 92   bkl
--   - Original version mostly by TAE

with TAE; use TAE;

--------------------------------------------------------------------
-- Event_Handlers
--------------------------------------------------------------------
-- Implementation Notes:
--   - This package contains the procedure that handle the events that the
--     user creates from working with the user interface. Each item on the
--     main and max panels that is designated to generate events has a
--     corresponding procedure.
--
--  The panel items:
--   Print_Button,    Refresh_Button, Capture_Button,
--   Zoom_Button,     Recenter_Button, Undo_Button,    Distance_Button,
--   Location_Button, Pan_Button,      Legend_Button,  Scroll_Up_Arrow,
--   Scroll_Dn_Arrow, Scroll_Rt_Arrow, Scroll_Hand,    Class_Banner,
--   Mini_Map_Xspace, Max_Button,      Remove_Sym_Butt, Critical_Alert,
--   Info_Alert,      Caution_Alert,   Cpa_Button,     Add_Symbol_Butt,
--   Filter_Button,   Scroll_Lt_Arrow, Map_Menu,       Symbol_Menu,
--   Session_Menu,    Main_Map_Xspace, Label1,         Label2,
--   Label3,          Label4,          Label5
--
-- Portability Issues:
--   - No operating system specific routines are called.
-- Anticipated Changes:
--------------------------------------------------------------------
package Event_Handlers is

  procedure Main_Map_Panel_Initialized
    ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

  procedure XWork_Space_Event
    ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

  procedure Max_Map_Panel_Initialized
    ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );
```

```
procedure Print_Button_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Refresh_Screen_Event;

procedure Capture_Button_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Zoom_Button_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Recenter_Button_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Undo_Button_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Distance_Button_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Location_Button_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Pan_Button_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Legend_Button_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Scroll_Up_Arrow_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Scroll_Dn_Arrow_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Scroll_Rt_Arrow_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Scroll_Hand_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Class_Banner_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Mini_Map_Xspace_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Max_Button_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );
```

```
procedure Remove_Sym_Butt_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Critical_Alert_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Info_Alert_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Caution_Alert_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Cpa_Button_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Add_Symbol_Butt_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Filter_Button_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Scroll_Lt_Arrow_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Map_Menu_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Symbol_Menu_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Session_Menu_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Main_Map_Xspace_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Label1_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Label2_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Label3_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

procedure Label4_Event
 ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );
 procedure Label5_Event
  ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );

end Event_Handlers;
```

C2 User Interface Library

```
-- File:    C2_UI_s.a
-- Author:  Bennett K. Larson (bkl)
-- System:  SparcStation 2, SunOS 4.1.2
-- Compiler: Verdix Ada Development System (VADS) 6.0
-- Revision History:
-- 11 Jun 92   bkl
--   - Final version
-- 25 May 92   bkl
--   - Original version

with C2_Shared_Lib;
with TAE;
-----------------------------------------------------------------------
-- C2_UI_Lib
-----------------------------------------------------------------------
-- Purpose:
--   This package encompasses the three user interface callable library
--   packages, Session, Map, and Symbol.
--
-- Effects:
--   - The expected usage is:
--     1. The user application is to "With in" this package to have visibility
--        to all needed functions associated with the C2 user interface.
-----------------------------------------------------------------------
package C2_UI_Lib is

  package Shared renames C2_Shared_Lib;

  Current_X_Event : TAE.Tae_Wpt.Wpt_Eventptr;

  ---------------------------------------------------------------------
  -- Set_Default_Display_ID
  ---------------------------------------------------------------------
  -- Purpose:
  --   This procedure is called by the Monitor Task to get the Display ID
  --   for later use with Xlib calls.
  -- Exceptions:
  --    None.
  ---------------------------------------------------------------------
  procedure Set_Default_Display_ID;
```

```
-------------------------------------------------------------------
-- Set_Main_Map_Panel_Info
-------------------------------------------------------------------
-- Purpose:
--   This procedure sets the Info value of the Main Map Panel for later use
--   by the Session, Map, and Symbol libraries.
-- Exceptions:
--    None.
-------------------------------------------------------------------
procedure Set_Main_Map_Panel_Info
        ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );


-------------------------------------------------------------------
-- Set_Max_Map_Panel_Info
-------------------------------------------------------------------
-- Purpose:
--   This procedure sets the Info value of the Max Map Panel for later use
--   by the Session, Map, and Symbol libraries.
-- Exceptions:
--    None.
-------------------------------------------------------------------
procedure Set_Max_Map_Panel_Info
        ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );



-------------------------------------------------------------------
package Session_UI_Lib is

 use Shared;


 -------------------------------------------------------------------
 -- Initialize Labels
 -------------------------------------------------------------------
 -- Purpose:
 --   This procedure intializes the labels after getting the label
 --   contents from the shared library.
 -- Exceptions:
 --    None.
 -------------------------------------------------------------------
 procedure Initialize_Labels;
```

```
-----------------------------------------------------------------
-- Display_Label
-----------------------------------------------------------------
-- Purpose:
--  This procedure displays information in one of the labels on the
--  user interface.
-- Exceptions:
--   None.
-----------------------------------------------------------------
procedure Display_Label
        ( Label_Position : in Integer );


-----------------------------------------------------------------
-- Change_Classification
-----------------------------------------------------------------
-- Purpose:
--  This procedure changes the classification of the banner on the main
--  map and the label on the max map.
-- Exceptions:
--   None.
-----------------------------------------------------------------
procedure Change_Classification
        ( Info : in TAE.Tae_Wpt.Event_Context_Ptr );


-----------------------------------------------------------------
-- Change_Info_Type
-----------------------------------------------------------------
-- Purpose:
--  This procedure changes the type of information being used, either
--  live or exercise.
-- Exceptions:
--   None.
-----------------------------------------------------------------
procedure Change_Info_Type;


-----------------------------------------------------------------
-- Display_Function_Message
-----------------------------------------------------------------
-- Purpose:
--  This procedure displays the function result messages at the
--  bottom of the main map panel.
-- Exceptions:
--   None.
-----------------------------------------------------------------
procedure Display_Function_Message;
```

```
 ------------------------------------------------------------------
 -- Display_Alert_Message
 ------------------------------------------------------------------
 -- Purpose:
 --   This procedure displays an alert message in one of the alert areas.
 -- Exceptions:
 --   None.
 ------------------------------------------------------------------
 procedure Display_Alert_Message;

 ------------------------------------------------------------------
 -- User_Quit_Interface
 ------------------------------------------------------------------
 -- Purpose:
 --   This procedure
 -- Exceptions:
 --   None.
 ------------------------------------------------------------------
 function User_Quit_Interface
          return Boolean;


 ------------------------------------------------------------------
 -- Quit_User_Interface
 ------------------------------------------------------------------
 -- Purpose:
 --   This procedure sets the Is_User_Interface_Done value to TRUE;.
 -- Exceptions:
 --   None.
 ------------------------------------------------------------------
 procedure Quit_User_Interface;

end Session_UI_Lib;


-------------------------------------------------------------------
package Map_UI_Lib is

 use Shared;


 ------------------------------------------------------------------
 -- Initialize_Map
 ------------------------------------------------------------------
 -- Purpose:
 --   This procedure
 -- Exceptions:
 --   None.
 ------------------------------------------------------------------
 procedure Initialize_Map;
```

```
---------------------------------------------------------------
-- Get_Map_Info
---------------------------------------------------------------
-- Purpose:
--  This procedure returns information about the map currently being
--  displayed. Currently there is no user interface facility to
--  implement this, but it could be implemented.
-- Exceptions:
--  None.
---------------------------------------------------------------
function Get_Map_Info
        return Map_Info_Type;

---------------------------------------------------------------
-- Change_Map_Name
---------------------------------------------------------------
-- Purpose:
--  This procedure specifies the file name of the map to display. This
--  is usually used only to give the name of the initial map.
-- Exceptions:
--  None.
---------------------------------------------------------------
procedure Change_Map_Name
        ( Map_Info : in Map_Info_Type );


---------------------------------------------------------------
-- Redraw_Main_Map
---------------------------------------------------------------
-- Purpose:
--  This procedure is called by an event handler after an expose event.
-- Exceptions:
--  None.
---------------------------------------------------------------
procedure Redraw_Main_Map;


---------------------------------------------------------------
-- Redraw_Max_Map
---------------------------------------------------------------
-- Purpose:
--  This procedure is called by an event handler after an expose event.
-- Exceptions:
--  None.
---------------------------------------------------------------
procedure Redraw_Max_Map;


---------------------------------------------------------------
-- Redraw_Mini_Map
```

```
-------------------------------------------------------------
-- Purpose:
--   This procedure is called by an event handler after an expose event.
-- Exceptions:
--   None.
-------------------------------------------------------------
procedure Redraw_Mini_Map;


-------------------------------------------------------------
-- Zoom_Area
-------------------------------------------------------------
-- Purpose:
--   This procedure displays a feedback box for the user to select an
--   area to zoom.
-- Exceptions:
--   None.
-------------------------------------------------------------
procedure Zoom_Area;


-------------------------------------------------------------
-- Zoom_Preset
-------------------------------------------------------------
-- Purpose:
--   This procedure zooms the map to a preset scale.
-- Exceptions:
--   None.
-------------------------------------------------------------
procedure Zoom_Preset;


-------------------------------------------------------------
-- Recenter_Map
-------------------------------------------------------------
-- Purpose:
--   This procedure recenters the map on a default center location.
-- Exceptions:
--   None.
-------------------------------------------------------------
procedure Recenter_Map;
```

```
    ----------------------------------------------------------------
    -- Undo_Last_Map_Operation
    ----------------------------------------------------------------
    -- Purpose:
    --   This procedure can undo the last map operation listed above.
    -- Exceptions:
    --   None.
    ----------------------------------------------------------------
    procedure Undo_Last_Map_Operation;


    ----------------------------------------------------------------
    -- Calculate_Distance
    ----------------------------------------------------------------
    -- Purpose:
    --   This procedure calculates the distance between two points on a map.
    -- Exceptions:
    --   None.
    ----------------------------------------------------------------
    procedure Calculate_Distance;


    ----------------------------------------------------------------
    -- Show_Coordinate_Location
    ----------------------------------------------------------------
    -- Purpose:
    --   This procedure shows the coordinates of a selected location.
    -- Exceptions:
    --   None.
    ----------------------------------------------------------------
    procedure Show_Coordinate_Location;


    ----------------------------------------------------------------
    -- Pan_Map
    ----------------------------------------------------------------
    -- Purpose:
    --   This procedure automatically scrolls the map based on the movement
    --   of object selected as the center.
    -- Exceptions:
    --   None.
    ----------------------------------------------------------------
    procedure Pan_Map;

end Map_UI_Lib;
```

```
----------------------------------------------------------------
package Symbol_UI_Lib is

 use Shared;

 ---------------------------------------------------------------
 -- Add_New_Track
 ---------------------------------------------------------------
 -- Purpose:
 --   This procedure adds a new user-entered track to the list of tracks.
 --   It will be displayed if it meets certain criteria.
 -- Exceptions:
 --   None.
 ---------------------------------------------------------------
 procedure Add_New_Track
         ( Track_Info : in Track_Info_Type );


 ---------------------------------------------------------------
 -- Delete_Track
 ---------------------------------------------------------------
 -- Purpose:
 --   This procedure deletes a track from the track list.
 -- Exceptions:
 --   None.
 ---------------------------------------------------------------
 procedure Delete_Track
         ( Track_ID : in Track_ID_Type );


 ---------------------------------------------------------------
 -- Get_Track_Info
 ---------------------------------------------------------------
 -- Purpose:
 --   This procedure gets information about a certain track.
 -- Exceptions:
 --   None.
 ---------------------------------------------------------------
 function Get_Track_Info
         ( Track_ID   : in Track_ID_Type)
          return Track_Info_Type;
```

```
------------------------------------------------------------
-- Update_Track_Info
------------------------------------------------------------
-- Purpose:
--   This procedure updates the information about a track.
-- Exceptions:
--   None.
------------------------------------------------------------
procedure Update_Track_Info
        ( Track_Info : in Track_Info_Type );


------------------------------------------------------------
-- Display_Track
------------------------------------------------------------
-- Purpose:
--   This procedure Displays a track on the map.
-- Exceptions:
--   None.
------------------------------------------------------------
procedure Display_Track
        ( Track_ID : in Track_ID_Type );


------------------------------------------------------------
-- Calculate_CPA
------------------------------------------------------------
-- Purpose:
--   This procedure calculates the Closest Point of Approach between
--   two selected symbols given the current heading a speed (if any).
-- Exceptions:
--   None.
------------------------------------------------------------
procedure Calculate_CPA;


------------------------------------------------------------
-- Filter_Symbols
------------------------------------------------------------
-- Purpose:
--   This procedure removes or adds symbols on the user interface, but
--   does not delete them from the shared object library.
-- Exceptions:
--   None.
------------------------------------------------------------
procedure Filter_Symbols;;
```

```
    -----------------------------------------------------------------
    -- Get_Legend_Info
    -----------------------------------------------------------------
    -- Purpose:
    --   This procedure gets the information that explains the symbology
    --   of the current application.
    -- Exceptions:
    --   None.
    -----------------------------------------------------------------
    procedure Get_Legend_Info;

  end Symbol_UI_Lib;

end C2_UI_Lib;
```

C2 Shared Persistent Objects Library

```
-- File:    C2_Shared_Lib_s.a
-- Author:  Bennett K. Larson (bkl)
-- System:  SparcStation 2, SunOS 4.1.2
-- Compiler: Verdix Ada Development System (VADS) 6.0
-- Revision History:
-- 15 Jun 92   bkl
--   - Final version
-- 25 May 92   bkl
--   - Original version


With Calendar_Utilities;
------------------------------------------------------------------
-- C2_Shared_Lib
------------------------------------------------------------------
-- Purpose:
--   This package contains shared types and objects for the application and UI
--   packages, Session, Map, and Symbol.
------------------------------------------------------------------
package C2_Shared_Lib is

  --------------------------------------------------------------
  -- Label Types
  --------------------------------------------------------------
  Max_Label_Contents_Length : constant Natural := 28;
  Max_Label_Positions       : constant Natural := 5;

  subtype Label_Contents_Type is String (1..Max_Label_Contents_Length);
  subtype Label_Position_Type is Natural range 1.. Max_Label_Positions;

  type Label_List_Type is array
       (1..Max_Label_Positions) of Label_Contents_Type;

  --------------------------------------------------------------
  -- Alert Message Types
  --------------------------------------------------------------
  Max_Alert_Number_Each_Category : constant Natural := 999;
  Alert_Text_Len                 : constant Integer := 20;
  type Alert_Category_Type is (CRITICAL, CAUTION, INFO);

  type Alert_Message_Type is
       record
         Number : Natural range 0..Max_Alert_Number_Each_Category;
         Text   : String (1..Alert_Text_Len)
                := (1..Alert_Text_Len => ' ');
         Alert_Category : Alert_Category_Type;
         Acknowledgement_Required : Boolean;
       end record;
```

```
type Alert_Acknowledgement_Type is
    record
      Number        : Natural range 0..Max_Alert_Number_Each_Category;
      Alert_Category : Alert_Category_Type;
    end record;


---------------------------------------------------------------
-- Track Types
---------------------------------------------------------------
Max_Track_ID : constant Positive := 10000;

subtype Track_ID_Type is Positive range 1..Max_Track_ID;

subtype Degree_Type is Natural range 0..359;
subtype Minute_Type is Natural range 0..59;
subtype Second_Type is Natural range 0..59;

subtype Range_Type  is Float range 0.0..Float'LAST;
subtype Speed_Type  is Float range 0.0..Float'LAST;
subtype Height_Type is Float;

subtype Zone_Type is Character range 'A'..'Z';

type Angle_Type is
    record
      Degree : Degree_Type;
      Minute : Minute_Type;
      Second : Second_Type;
    end record;

type Latitude_Hemisphere_Type  is (NORTH, SOUTH);
type Longitude_Hemisphere_Type is (EAST, WEST);

type Latitude_Type is
    record
      Angle      : Angle_Type;
      Hemisphere : Latitude_Hemisphere_Type;
    end record;

type Longitude_Type is
    record
      Angle      : Angle_Type;
      Hemisphere : Longitude_Hemisphere_Type;
    end record;
```

```
type Position_Type is
    record
     Latitude  : Latitude_Type;
     Longitude : Longitude_Type;
    end record;

type Position_Time_Type is
    record
     The_Time : Calendar_Utilities.Time;
     Zone     : Zone_Type;
    end record;


type North_Type is (TRUE, MAGNETIC);

type Bearing_Type is
    record
     Direction_Angle : Angle_Type;
     Reference_North : North_Type;
    end record;


type Relative_Position_Type is
    record
     Bearing   : Bearing_Type;
     The_Range : Range_Type;
    end record;

type Origin_Type is (REMOTE, LOCAL_AUTO, LOCAL_MANUAL);
type Identity_Type is (UNKNOWN, FRIENDLY, HOSTILE);

type Track_Info_Type is
    record
     Track_ID          : Track_ID_Type;
     Position          : Position_Type;
     Position_Time     : Position_Time_Type;
     Relative_Position : Relative_Position_Type;
     Origin            : Origin_Type;
     Course            : Angle_Type;
     Speed             : Speed_Type;
     Identity          : Identity_Type;
     Height            : Height_Type;
    end record;
```

```
-------------------------------------------------------------------
-- Map Types
-------------------------------------------------------------------
Max_File_Name_Len : constant Integer := 256;

subtype Map_Name_Type is String (1..Max_File_Name_Len);

type Map_Info_Type is
    record
      Map_File_Name : Map_Name_Type := (1..Max_File_Name_Len => ' ');
      Map_File_Name_Len : Natural;
    end record;

-------------------------------------------------------------------
package Shared_Session_Lib is



 ------------------------------------------------------------------
 -- User_Interface_Done
 ------------------------------------------------------------------
 -- Purpose:
 --   This procedure checks whether or not the user has quit the
 --   user interface.
 -- Exceptions:
 --   None.
 ------------------------------------------------------------------
 function User_Interface_Done
          return Boolean;


 ------------------------------------------------------------------
 -- Set_User_Interface_Done
 ------------------------------------------------------------------
 -- Purpose:
 --   This procedure sets the Is_User_Interface_Done value to TRUE;.
 -- Exceptions:
 --   None.
 ------------------------------------------------------------------
 procedure Set_User_Interface_Done;
```

```
-----------------------------------------------------------------
-- Set_All_Labels
-----------------------------------------------------------------
-- Purpose:
--   This procedure sets all of the labels at once.
-- Exceptions:
--   None.
-----------------------------------------------------------------
procedure Set_All_Labels
        ( Label_List : in Label_List_Type );


-----------------------------------------------------------------
-- Get_All_Labels
-----------------------------------------------------------------
-- Purpose:
--   This functions returns all of the labels at once.
-- Exceptions:
--   None.
-----------------------------------------------------------------
function Get_All_Labels
        return Label_List_Type;


-----------------------------------------------------------------
-- Set_Label
-----------------------------------------------------------------
-- Purpose:
--   This procedure sets the contents of only one label.
-- Exceptions:
--   None.
-----------------------------------------------------------------
procedure Set_Label
        ( Label_Contents : in Label_Contents_Type;
          Label_Position : in Label_Position_Type );


-----------------------------------------------------------------
-- Get_Label
-----------------------------------------------------------------
-- Purpose:
--   This function returns the contents of only one label.
-- Exceptions:
--   None.
-----------------------------------------------------------------
function Get_Label
        ( Label_Position : in Label_Position_Type )
        return Label_Contents_Type;

end Shared_Session_Lib;
```

```
-----------------------------------------------------------------
package Shared_Map_Lib is


 ----------------------------------------------------------------
 -- Set_Map_Info
 ----------------------------------------------------------------
 -- Purpose:
 --   This procedure sets the map information info.
 -- Exceptions:
 --   None.
 ----------------------------------------------------------------
 procedure Set_Map_Info
          ( Map_Info : in Map_Info_Type );


 ----------------------------------------------------------------
 -- Get_Map_Info
 ----------------------------------------------------------------
 -- Purpose:
 --   This function gets the map information values.
 -- Exceptions:
 --   None.
 ----------------------------------------------------------------
 function Get_Map_Info return Map_Info_Type;


end Shared_Map_Lib;


-----------------------------------------------------------------
package Shared_Symbol_Lib is



 ----------------------------------------------------------------
 -- Add_New_Track
 ----------------------------------------------------------------
 -- Purpose:
 --   This procedure adds a new track object.
 -- Exceptions:
 --   None.
 ----------------------------------------------------------------
 procedure Add_New_Track
          ( Track_Info : Track_Info_Type );
```

```
    -----------------------------------------------------------------
    -- Delete_Track
    -----------------------------------------------------------------
    -- Purpose:
    --   This procedure deleted a track object.
    -- Exceptions:
    --   None.
    -----------------------------------------------------------------
    procedure Delete_Track
          ( Track_ID : in Track_ID_Type );


    -----------------------------------------------------------------
    -- Get_Track_Info
    -----------------------------------------------------------------
    -- Purpose:
    --   This procedure gets information on the specified track.
    -- Exceptions:
    --   None.
    -----------------------------------------------------------------
    procedure Get_Track_Info
          ( Track_ID   : in  Track_ID_Type;
            Track_Info : out Track_Info_Type );

    -----------------------------------------------------------------
    -- Update_Track_Info
    -----------------------------------------------------------------
    -- Purpose:
    --   This procedure modifies a current track's information
    -- Exceptions:
    --   None.
    -----------------------------------------------------------------
    procedure Update_Track_Info
          ( Track_Info : in Track_Info_Type );

  end Shared_Symbol_Lib;

end C2_Shared_Lib;
```

Classic-Ada Track Objects

```
-- File:     display_obj_s.ca  (Classic-Ada)
-- Author:   Bennett K. Larson (bkl)
-- System:   SparcStation 2, SunOS 4.1.2
-- Pre-Processor: Classic-Ada (1989)
-- Compiler: Verdix Ada Development System (VADS) 6.0

class Display_Object is

  instance method Set_Obj_Number ( Object_Number : in Integer );
  instance method Get_Obj_Number ( Object_Number : out Integer );

end Display_Object;
```

```
-- File:     display_obj_b.ca  (Classic-Ada)
-- Author:   Bennett K. Larson (bkl)
-- System:   SparcStation 2, SunOS 4.1.2
-- Pre-Processor: Classic-Ada (1989)
-- Compiler: Verdix Ada Development System (VADS) 6.0

with Text_IO;
use  Text_IO;

class body Display_Object is

 Display_Object_Number : instance Integer;


 -------------------------------------------------------------------
 instance method Set_Obj_Number ( Object_Number : in Integer ) is

 begin

  Display_Object_Number := Object_Number;
  New_Line;
  Put_Line ("DISPLAY OBJ: Display Object Number Set.");

 end Set_Obj_Number;


 -------------------------------------------------------------------
 instance method Get_Obj_Number ( Object_Number : out Integer ) is

 begin

  Object_Number := Display_Object_Number;
  New_Line;
  Put_Line ("DISPLAY OBJ: Display Object Number Retrieved.");

 end Get_Obj_Number;


end Display_Object;
```

```
-- File:    track_s.ca   (Classic-Ada)
-- Author:   Bennett K. Larson (bkl)
-- System:   SparcStation 2, SunOS 4.1.2
-- Pre-Processor: Classic-Ada (1989)
-- Compiler: Verdix Ada Development System (VADS) 6.0

class Track is

  superclass Display_Object;

  instance method Set_Speed ( Current_Speed : in  Float );
  instance method Get_Speed ( Current_Speed : out Float );

end Track;
```

```
-- File:    track_b.ca  (Classic-Ada)
-- Author:  Bennett K. Larson (bkl)
-- System:  SparcStation 2, SunOS 4.1.2
-- Pre-Processor: Classic-Ada (1989)
-- Compiler: Verdix Ada Development System (VADS) 6.0

with Text_IO;
use  Text_IO;

class body Track is

  Speed    : Instance Float;


  ----------------------------------------------------------------------
  instance method Set_Speed ( Current_Speed: in Float ) is

  begin

    Speed := Current_Speed;
    New_Line;
    Put_Line ("TRACK: Track Speed Set.");


  end Set_Speed;

  ----------------------------------------------------------------------
  instance method Get_Speed ( Current_Speed: out Float ) is

  begin

    Current_Speed := Speed;
    New_Line;
    Put_Line ("TRACK: Track Speed Retrieved.");


  end Get_Speed;


end Track;
```

```
-- File:     air_track_s.ca  (Classic-Ada)
-- Author:   Bennett K. Larson (bkl)
-- System:   SparcStation 2, SunOS 4.1.2
-- Pre-Processor: Classic-Ada (1989)
-- Compiler: Verdix Ada Development System (VADS) 6.0

class Air_Track is

  superclass Track;


  method Create ( New_Air_Track : out Object_ID );
  instance method Delete;

  instance method Set_Altitude ( Current_Altitude : in  Float );
  instance method Get_Altitude ( Current_Altitude : out Float );

end Air_Track;
```

```
-- File:    air_track_b.ca  (Classic-Ada)
-- Author:  Bennett K. Larson (bkl)
-- System:  SparcStation 2, SunOS 4.1.2
-- Pre-Processor: Classic-Ada (1989)
-- Compiler: Verdix Ada Development System (VADS) 6.0

with Text_IO;
use  Text_IO;

class body Air_Track is

 Altitude   : Instance Float;

 ------------------------------------------------------------------
 method Create (New_Air_Track: out Object_ID) is

 begin
  New_Air_Track := Instantiate;
  New_Line;
  Put_Line ("AIR TRACK: Air Track Created!!");
 end Create;


 ------------------------------------------------------------------
 instance method Delete is

 begin
  New_Line;
  Put_Line("AIR TRACK: Air Track Deleted");
  destroy;
 end Delete;


 ------------------------------------------------------------------
 instance method Set_Altitude ( Current_Altitude: in Float ) is

 begin

  Altitude := Current_Altitude;
  New_Line;
  Put_Line ("AIR TRACK: Air Track Altitude Set.");

 end Set_Altitude;

 ------------------------------------------------------------------
 instance method Get_Altitude ( Current_Altitude: out Float ) is

 begin

  Current_Altitude := Altitude;
```

```
    New_Line;
    Put_Line ("AIR TRACK: Air Track Altitude Retrieved.");

  end Get_Altitude;

end Air_Track;
```

Track Object Package Specification

```
-- File:    Track_Lib_s.a  (No Classic-Ada needed here)
-- Author:  Bennett K. Larson (bkl)
-- System:  SparcStation 2, SunOS 4.1.2
-- Compiler: Verdix Ada Development System (VADS) 6.0


with C2_Shared_Lib; use C2_Shared_Lib;
------------------------------------------------------------------
-- Track_Lib
------------------------------------------------------------------
-- Purpose:
--   This package manages the air tracks
------------------------------------------------------------------
package Track_Lib is


  ----------------------------------------------------------------
  -- Instantiate_Track
  ----------------------------------------------------------------
  -- Purpose:
  --   This procedure creates a new track object.
  -- Exceptions:
  --   None.
  ----------------------------------------------------------------
  function Instantiate_Track
          ( Track_Info : in Track_Info_Type ) return Integer;


  ----------------------------------------------------------------
  -- Set_Track_Data
  ----------------------------------------------------------------
  -- Purpose:
  --   This procedure updates the given track with the given info.
  -- Exceptions:
  --   None.
  ----------------------------------------------------------------
  procedure Set_Track_Data ( Info : in Track_Info_Type );


  ----------------------------------------------------------------
  -- Get_Track_Data
  ----------------------------------------------------------------
  -- Purpose:
  --   This procedure returns the track info.
  -- Exceptions:
  --   None.
  ----------------------------------------------------------------
  function Get_Track_Data ( ID : in Track_ID_Type ) return Track_Info_Type;
```

```
   ------------------------------------------------------------------
   -- Delete_Track
   ------------------------------------------------------------------
   -- Purpose:
   --   This procedure deletes a given track.
   -- Exceptions:
   --   None.
   ------------------------------------------------------------------
   procedure Delete_Track ( ID : in Track_ID_Type );


end Track_Lib;
```

```
-- File:    Track_Lib_b.ca  (Classic-Ada)
-- Author:   Bennett K. Larson (bkl)
-- System:   SparcStation 2, SunOS 4.1.2
-- Preprocessor : Classic-Ada 1989
-- Compiler: Verdix Ada Development System (VADS) 6.0


with String_Utilities; use String_Utilities;
with Character_Utilities; use Character_Utilities;
with List_Single_Unbounded_Managed;
with List_Utilities_Single;

with Text_IO;
use  Text_IO;
with Air_Track;
use  Air_Track;

-----------------------------------------------------------------------
-- Track_Lib (Track Objects)
-----------------------------------------------------------------------
-- Implementation Notes:
--   - This package manages the object-oriented tracks using Classic-Ada
--     send messages to create objects, set object data, get object data, and
--     delete objects.
--   - The Booch list components are used to manage a linked list of
--     object IDs that are created by the Classic-Ada code and the
--     corresponding sequential track IDs that are used to keep
--     track of the air tracks outside of this package.
-- Anticipated Changes:
--   - The list only contains air tracks now.  Enhancements to the code
--     would allow surface tracks to be maintained as well.
-----------------------------------------------------------------------
package body Track_Lib is

 package Int_IO is new Integer_IO (Integer);
 use Int_IO;

 package Flt_IO is new Float_IO (Float);
 use Flt_IO;


 Next_Track_ID : Integer := 1;

 type List_Node_Type is
  record
   Track_ID       : Integer;
   Object_Pointer : Object_ID;
  end record;

 List_Node : List_Node_Type;
```

```
package Air_Track_List_Package is new
 List_Single_Unbounded_Managed ( List_Node_Type );
use Air_Track_List_Package;

Air_Track_List : Air_Track_List_Package.List;

Package Air_Track_List_Utilities is new
 List_Utilities_Single ( Item      => List_Node_Type,
                         List      => Air_Track_List_Package.List,
                         Clear     => Air_Track_List_Package.Clear,
                         Construct => Air_Track_List_Package.Construct,
                         Swap_Tail => Air_Track_List_Package.Swap_Tail,
                         Is_Null   => Air_Track_List_Package.Is_Null,
                         Tail_Of   => Air_Track_List_Package.Tail_Of );

use Air_Track_List_Utilities;

-- Air_Track_Object : Object_ID := Air_Track.class_object;


-------------------------------------------------------------------
-- Find_Node
-------------------------------------------------------------------
-- Implementation Notes:
--   - This routine finds the track node to manipulate.
--   - Note: No error checking if node not found.
-------------------------------------------------------------------
function Find_Node ( ID : Integer ) return List_Node_Type is

  Temp_Air_Track_List : Air_Track_List_Package.List := Air_Track_List;
  Current_List_Node   : List_Node_Type;

begin

  while not Is_Null ( Temp_Air_Track_List ) loop

    Current_List_Node := Head_Of ( Temp_Air_Track_List );

    if ID = Current_List_Node.Track_ID then
      return Current_List_Node;
    end if;

    -- get next node
    Temp_Air_Track_List := Tail_Of ( Temp_Air_Track_List );

  end loop;

  New_Line;
  Put_Line ("Find_Node: Node Not Found!");
```

```
end Find_Node;


-------------------------------------------------------------------
-- Set_Track_Data
-------------------------------------------------------------------
-- Implementation Notes:
--   - This routine only sets track speed for now.
-------------------------------------------------------------------
procedure Set_Track_Data ( Info : in Track_Info_Type ) is

  Current_List_Node   : List_Node_Type;

begin

  Current_List_Node := Find_Node ( Info.Track_ID );

  -- test inherited set_speed method
  New_Line;
  Put_Line ("Sending Air Track a msg to update its speed");
  send ( Current_List_Node.Object_Pointer, Set_Speed,
       Current_Speed => Info.Speed );

end Set_Track_Data;


-------------------------------------------------------------------
-- Instantiate_Track
-------------------------------------------------------------------
-- Implementation Notes:
--   - This routine creates an air track and returns a track ID for
--     external use.
-------------------------------------------------------------------
function Instantiate_Track
        ( Track_Info : in Track_Info_Type ) return Integer is

  Current_Track_ID : Integer;
  New_Air_Track_Object : Object_ID := Air_Track.class_object;

begin


  -- test create method
  New_Line;
  Put_Line ( "Sending Air Track a msg to create itself." );
  send ( New_Air_Track_Object, Create,
       New_Air_Track => New_Air_Track_Object );

  -- Track_Info.Track_ID should = Next_Track_ID, not checked yet, though
  List_Node.Track_ID       := Next_Track_ID;
```

```
 List_Node.Object_Pointer := New_Air_Track_Object;

 -- test inherited set_obj_number method
 New_Line;
 Put_Line ( "Sending Air Track a msg to set display object number." );
 send ( List_Node.Object_Pointer, Set_Obj_Number,
      Object_Number => Next_Track_ID );

 -- test inherited set_speed method
 New_Line;
 Put_Line ("Sending Air Track a msg to initialize its speed");
 send ( List_Node.Object_Pointer, Set_Speed,
      Current_Speed => Track_Info.Speed );


 -- Insert does not work on a null list, so must use construct if empty list
 if Next_Track_ID /= 1 then
  Insert ( List_Node, Air_Track_List );
 else
  Construct ( List_Node, Air_Track_List );
 end if;

 Current_Track_ID := Next_Track_ID;

 Next_Track_ID := Next_Track_ID + 1;

 return Current_Track_ID;

end Instantiate_Track;


--------------------------------------------------------------------
-- Get_Track_Data
--------------------------------------------------------------------
-- Implementation Notes:
--   - This routine only gets track speed for now.
--------------------------------------------------------------------
function Get_Track_Data ( ID : in Track_ID_Type ) return Track_Info_Type is

 Track_Info : Track_Info_Type;
 Current_List_Node   : List_Node_Type;

begin

 Current_List_Node := Find_Node ( ID );

 -- test inherited get_speed method
 New_Line;
 Put_Line ("Sending Air Track a msg to get its speed");
 send (Current_List_Node.Object_Pointer, Get_Speed,
      Current_Speed => Track_Info.Speed );
  -- test inherited get_obj_number method
  send (Current_List_Node.Object_Pointer, Get_Obj_Number,
      Object_Number => Track_Info.Track_ID );

  return Track_Info;

 end Get_Track_Data;


 -----------------------------------------------------------------------
 -- Delete_Track
 -----------------------------------------------------------------------
 -- Implementation Notes:
 --   - This routine deletes a track. Does not delete list nodes, though.
 -----------------------------------------------------------------------
 procedure Delete_Track ( ID : in Track_ID_Type ) is

  Current_List_Node   : List_Node_Type;

 begin

  Current_List_Node := Find_Node ( ID );

  New_Line;
  Put_Line ("TRACK LIB: Sending Air Track a msg to delete itself");
  send (Current_List_Node.Object_Pointer, Delete);

  Next_Track_ID := Next_Track_ID - 1;

 end Delete_Track;


end Track_Lib;
```

APPENDIX C

ON-LINE AND OTHER ADA AND MAP SOURCES

This Appendix lists sources of Ada and map information that is available on the Internet or by regular mail or telephone. The categories of the sources are Ada Information, Ada Map Function Information, Map Data Information, and General Information.

Many of the sources offer unrestricted access to their information. Some of the sources not available on-line are in the process of connecting to the network. Other sources are intentionally off-line to restrict access to controlled information. Some sources are on-line and restrict access as well.

On-line resources are one of the easiest and quickest ways to get Ada or map information. Most of the sources listed in this appendix allow anonymous and/or free access to large amounts of information. Most of this access is allowed through the Internet using the File Transfer Protocol (FTP). Some sites provide bulletin board (BBS) services using Telnet on the Internet or modem dial-up.

The following information may become outdated rather quickly, but is offered as a sampling of what kind of sources exist. For a fairly recent and comprehensive listing of Ada resources, see [40]. However, according to the author of this source, Karl Nyberg, a 1992 and even a 1993 edition of this catalog will probably not be published. Another good source for Ada and software reuse information is a recent Ada Letters article [33]. The Ada Information Clearinghouse (AdaIC) described below is probably the best source for current Ada information.

1. Ada Information

a. Ada Joint Program Office

According to the AdaIC, "the Ada Joint Program Office (AJPO) is responsible for informing the (military) community about Ada, facilitating the language's implementation in the (military) Services, and maintaining the integrity of he language." [5] The AdaIC is the information source for the AJPO.

(1) Address: Ada Joint Program Office, Room 3E118, The Pentagon, Washington, D.C. 20301-3081

b. Ada Information Clearinghouse

The Ada Information Clearinghouse (AdaIC) is an excellent source for information about Ada products, tools. and published articles. The IIT Research Institute operates the AdaIC for the AJPO. The AdaIC maintains on-line information files, publishes a quarterly newsletter, and in general provides a full spectrum of information on the current status of Ada.

(1) Address: Ada Information Clearinghouse, c/o IIT Research Institute, 4600 Forbes Boulevard Lanham, MD 20706-4320

(2) Telephone/Fax: (703) 685-1477, (800) AdaIC-11, FAX (703) 685-7019

(3) E-mail:

- Internet: adainfo@ajpo.sei.cmu.edu
- CompuServe: 70312,3303

(4) BBS: (703) 614-0215, DSN: 224-0215

(5) Internet (FTP): ajpo.sei.cmu.edu

c. Ada Software Repository

The Ada software repository (ASR) is probably the first government site to start collecting Ada software source code and general-use computer programs for the DOD. For a complete history see [15]. This reference is somewhat dated, but it is still a good general introduction to the repository.

(1) Address: LC-38, Building 23640, White Sands Missile Range, NM 88002

(2) Telephone: (505) 678-9430, DSN: 258-9430

(3) E-mail: wancho@wsmr-simtel20.army.mil

(4) DDN (FTP): wsmr-simtel20.army.mil

(5) Internet (FTP): archive.wustl.edu

d. Software Technology Support Center

According to a recent a recent Software Technology Support Center (STSC) newsletter, the STSC "is the (Air Force) central focal point for technical expertise and management support of software tools, methods, and environments." [52] The newsletter, called CrossTalk, is available free to any individuals actively involved in the military software development process. Each April the STSC sponsors a software development and maintenance meeting in Salt Lake City called the Software Technology Conference. The STSC maintains a BBS that should be on the Internet by the end of 1992.

(1) Address: OOALC/TISAC, Hill AFB, Utah 84056
(2) Telephone: (801) 777-7703, DSN: 458-7703
(3) E-mail (Customer Service): wrightr@oodis01.af.mil
(4) BBS: (801) 777-7553, DSN: 458-7553

e. *Software Technology for Adaptable, Reliable Systems*

The Software Technology for Adaptable, Reliable Systems (STARS) was chartered for the purpose of achieving dramatic improvements in software productivity while continuing to make incremental improvements in quality and reliability. The software developed under the STARS program is now maintained by the ASSET effort described below.

(1) Address: STARS Technology Center, 801 North Randolph Street, Suite 400, Arlington, VA 22203
(2) Telephone: (703) 243-8655
(3) E-mail: danglert@source.asset.com
(4) BBS: (304) 594-3635
(5) Internet (Telnet): source.asset.com
- Login ID: starsbbs
- On-line registration supported

(6) Internet (FTP): source.asset.com

f. *Asset Source for Software Engineering Technology*

The DOD established the Asset Source for Software Engineering Technology (ASSET) under the STARS program to provide a more comprehensive source for reusable software. A key resource found on the internet host is an on-line newsletter containing recent reuse information from all over the Internet.

(1) Address: 2611 Cranberry Square, Morgantown, WV 26505
(2) Telephone: (304) 594-1762
(3) E-mail: danglert@source.asset.com
(4) Internet (Telnet): source.asset.com
- An account can be established by postal mail only

(5) Internet (FTP): source.asset.com

g. NASA

The TAE Plus user interface management tool described in this thesis was created by NASA and is supported via the Internet. The TAE Plus support office maintains a list of files that users have submitted as example of how they have used TAE Plus for a variety of applications. While configuring and writing the software for this thesis, a few of the examples were valuable in helping to understand how to use certain TAE Plus capabilities. The file list also contains updates and fixes to problems found in TAE Plus.

The TAE Plus tool is distributed by the Computer Software Management and Information Center (COSMIC) at the University of Georgia. The address for COSMIC is listed below and is followed by the support office information.

(1) Address: COSMIC, The University of Georgia, 382 East Broad Street, Athens, GA 30602.

(2) Telephone: (404) 542-3265

(3) Telex: 490 999 1619

The support office can be contacted at:

(4) Address: Goddard Space Flight Center, TAE Support Office, Code 522, Greenbelt, MD 20771.

(5) Telephone: (301) 286-6034, FTS 888-6034

(6) E-mail: taeso@postman.gsfc.nasa.gov

(7) Internet (FTP): kong.gsfc.nasa.gov (128.183.94.10)

2. Ada Map Function Information

a. Air Force Rome Laboratory

The Rome Lab has what is called the Common Mapping System (CMS) which is a set of Ada software components which do map data transformations. The Ada software is owned by the government. Information can be obtained from Jeff Hansen (Rome Labs/IRD) at:

(1) Rome Labs, Griffiss AFB, NY.

(2) Telephone: (315) 330-7889

3. Map Data Information

a. Defense Mapping Agency

The Defense Mapping Agency (DMA) distributes map data in many different formats and data processing software and on a variety of media. Media types include tape, CD-

ROM, and hardcopy. Files can also be transferred over the Worldwide Military Command and Control System (WWMCCS) network. Although the DMA primarily provides data to military users, non-military users can obtain limited data sets of certain formats. A catalog describing all of this information in detail is available [18].

(1) Address: Director, Defense Mapping Agency, Attn: PR, 8613 Lee Highway, Fairfax, VA, 22031-2137

(2) Telephone: (800) 826-0342, (301) 227-2495, DSN: 287-2495

b. *Earth Resources Data Center*

The Earth Resources (EROS) data center is a source for both WDB-II map data and Landsat images. The WDB data has been restructured into a Relational WDB II (RWDB2) format and a tape of the 450MB of data costs $320. See the following Grebyn source for a better price. EROS is located in Sioux Falls, SD, and can be reached at:

(1) Address: Department of Interior, Sioux Falls, SD, 57198.

(2) Telephone: (605) 594-6507

c. *United States Geological Service*

The United States Geological Service (USGS) is a source for a wide variety of data. Mr. Warren Tucker, from Tuckerware, provided this information: "The USGS has a $29 CD-ROM with 561 Megabytes of data, ranging from useless/difficult to exceptionally valuable. It is a bit out of date, but if you take proposed highways as being real ones, you'll get a pretty good picture." [62] The CD-ROM and the original WDB-II data and also the RWDB2 data can be obtained from:

(1) Address: Earth Science Information Center, U.S. Geological Survey, 507 National Center, Reston, VA 22092.

(2) Telephone: (703) 648-6045, (800) USA-MAPS

d. *Grebyn Corporation*

This company does work with RWDB2 data and a tape of it can be purchased from this company for $60. According to Grebyn, the RWDB2 data may soon be available on-line from some Internet host. Grebyn can be contacted at:

(1) Address: Post Office Box 497, Grebyn Corporation, Vienna, VA 22183-0497.

(2) Telephone: (703) 281-2194

e. Internet Sites

Several host computers on the Internet have collections of map data. Not much of it is well organized, but it is free for the taking. Most of it is not in the original WDB-II format. Some programs do exist that read and view the data. Files can be obtained using FTP from the following sites:

(1) hanauma.stanford.edu: This site has WDB-II data that was derived from the original CIA data. However, this site may not exist much longer because of hardware support problems. A new site is being considered.

(2) spectrum.xerox.com: This site has USGS DEM (Digital Elevation Model) and Digital Line Graph (DLG) data. There is also geographic names information that goes with the map data. This site also has some CIA WDB-II derived data and Census Bureau TIGER/Line map data. The usefulness of the USGS and Census data for C^2 research purposes could not be determined from these sources.

(3) pil.arc.umn.edu: A program called mfil on this host can decode the WDB-II data.

(4) gatekeeper.dec.com: This host has a data set that was derived from the original WDB-II data and is available on-line. There are also some ready-to-print PostScript files available. The original WDB-II data can be obtained by sending a tape to the system administrator. The original data was obtained from the National Technical Information Service (NTIS).

(5) alum.wr.usgs.gov: The data format of the maps at this site are unknown.

4. General Information

Sources in this section provide some ways to access widely available software and other information.

a. Usenet Newsgroups

An informal, but very useful, source of information is Usenet. Usenet is a group of people who exchange articles tagged with one or more universally-recognized labels, called "newsgroups." The articles are forwarded through the Internet to participating sites, world-wide. Articles may be messages or source code. The number of newsgroups is large and topics vary widely. The following newsgroups provided helpful information for this thesis.

(1) **comp.lang.ada**: for Ada information and programming help.

(2) **comp.software-eng**: for design method information.

(3) **comp.windows.x**: for x programming tips.

(4) **comp.sources.unix**: for example map programs.

(5) **sci.geo.geology**: for map data information.

(6) **comp.internet.library**: for literature search host information.

(7) **comp.graphics**: for map and algorithm information.

b. The Archie (Archive) Service

Archie (archive without the 'v') servers are computer systems that allow anonymous Internet Telnet (and e-mail) access for searching a large database of almost every known piece of free software on the Internet. These sites are mainly run by academic institutions and submitting files for the database is voluntary. Still, there is probably no better place to start looking for free Ada or map information than at an Archie site. One nice feature of the Archie software is that the result of a search can be e-mailed back to the searcher.

The Archie service currently has over 50 Gigabytes of information and it is automatically updated once a month by the sites that are configured with the client software. Archie also has a "whatis" database that contains a brief synopsis for over 3,500 public domain software packages. for a variety of application areas. More information is continually being added.

(1) Address: UNIX Support Group, Computing Centre, McGill University, room 200, Burnside Hall, 805 Sherbrooke Street West, Montreal, Quebec, Canada H3A 2K6.

(2) Telephone: (514) 398-3709

(3) E-mail (general information/help): archie-l@archie.mcgill.ca

(4) E-mail access: archie@archie.mcgill.ca. To get access to Archie if e-mail access is the only means available, send a message to the address given with only the work HELP in the text area. The Archie e-mail server will return the necessary information.

(5) U.S. Internet sites:

- archie.sura.net (128.167.254.179), SURAnet, College Park, MD.
- archie.ans.net (147.225.1.2), ANS, NY.
- archie.unl.edu (129.93.1.14), Lincoln, NE.
- archie.rutgers.edu (128.6.18.15),Piscataway, NJ.

LIST OF REFERENCES

1. Ada Information Clearinghouse, *Ada Quality and Style: Guidelines for Professional Programmers*, Version 2.00.02, Software Productivity Consortium, 1991.
2. Ada Information Clearinghouse, *Ada 9X Project Report: Ada 9X Requirements*, Ada Joint Program Office, Under Secretary of Defense for Acquisition, December 1990.
3. Ada Information Clearinghouse, *Overview of U.S. Air Force Report, Ada and C++: A Business Case Analysis*, Press Conference held by L. R. Mosemann, II, Deputy Assistant Secretary of the Air Force (Communications, Computers, and Logistics), 9 July 1991.
4. Ada Information Clearinghouse, *Ada Usage Database*, Ada Joint Program Office, Under Secretary of Defense for Acquisition, 1992.
5. Ada Information Clearinghouse, *On-line AdaIC Information File*, Ada Joint Program Office, Under Secretary of Defense for Acquisition, 1992.
6. Ada Information Clearinghouse, *Available Ada Bindings*, Ada Joint Program Office, Under Secretary of Defense for Acquisition, January 1992.
7. Andriole, S. J., *Command and Control Information Systems Engineering: Progress and Prospects*, Advances in Computers, Vol. 31, Edited by M. C. Yovits, Academic Press, 1990.
8. Andriole, S. J., *Storyboard Prototyping: A New Approach to Requirements Analysis*, QED Information Sciences, Inc., 1989.
9. Anderson, E. S., Functional Specification for a Generic C3I Station, M.S. Thesis, Naval Postgraduate School, Monterey, California, September 1990.
10. Beam, W. R., *Command, Control, and Communications Systems Engineering*, McGraw-Hill, 1989.
11. Booch, G., *Software Engineering with Ada*, 2d ed., Benjamin/Cummings, 1987.
12. Booch, G., *Software Components with Ada*, Benjamin/Cummings, 1987.
13. Booch, G., *Object-Oriented Design with Applications*, Benjamin/Cummings, 1991.
14. Buhr, R. J. A., *System Design with Ada*, Prentice-Hall, 1984.
15. Conn, R., *The Ada Software Repository and the Defense Data Network: A Resource Handbook*, New York Zoetrope, 1987.
16. Defense Advance Research Projects Agency, News Release, *Asset Source for Software Engineering Technology*, 3 December 1991.
17. Defense Mapping Agency, Products Catalog, *Digitizing The Future*, 2d ed., Department of Defense, October 1988.
18. Defense Mapping Agency, Products Catalog, *Digitizing The Future*, 3d ed., Department of Defense, No Date.

19. Deitel, H. M., *An Introduction to Operating Systems*, 2d ed., Addison-Wesley, 1990.

20. Department of Defense, *Joint Warfare of the US Armed Forces (Joint Pub 1)*, Government Printing Office, 11 November 1991.

21. de Paula, E. G., and Nelson, M. L., "Designing a Class Hierarchy," *Proceedings of Technology of Object-Oriented Languages and Systems (TOOLS) USA '91*, pp. 203-218, 31 July 1991.

22. Force Structure, Resource, and Assessment Directorate (J-8), Joint Staff, The Pentagon, Joint Theater Level Simulation (JTLS) Users' Guide, January 1991.

23. Frakes, W. B., and others, "Panel 1: Is Software Reuse Delivering?", *Proceedings of the 1991 IEEE 13th International Conference On Software Engineering*, pp. 52-59, IEEE Computer Society Press, May 1991.

24. Frame Technology, Corp., *Using FrameMaker*, X Window Version, San Jose, California, 1991.

25. Howard, T. L. J., and others, *A Practical Introduction to PHIGS and PHIGS Plus*, Addison-Wesley, 1991.

26. Interactive Development Environments, *Software Through Pictures Reference Manual*, version 4.2B, San Francisco, California, December 1990.

27. Interview between S. Ostermeier, Lieutenant Commander, USN (Ret.), Monterey, California, and the author, 26 January 1992.

28. Joint Chiefs of Staff, *Command and Control Functional Analysis and Consolidation Review Panel Report*, For Official Use Only, Department of Defense, 30 October 1991.

29. Joint Chiefs of Staff, *Command, Control, Communications, Computers, and Intelligence (C4I) for the Warrior*, briefing given by Col. Bryan, USA, (J6), 6 February 1992.

30. Joint Data Systems Support Center, Technical Memorandum TM 406-91, *Operational Concept Document of the Mapping and Graphic Information Capability (MAGIC)*, 15 January 1991.

31. Jim, A. S. and Guenter P. S., *Requirements Analysis for a Low Cost Combat Direction System (LCCDS)*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1990.

32. Krueger, C. W., "Software Reuse," To be Published in the Association for Computing Machinery's Computing Surveys, June 1992.

33. Levine, T., "Reusable Software Components," *Ada Letters*, v. XI, pp. 66-71, May/June 1991.

34. Mark V Systems Limited, *ObjectMaker User's Manual*, Version 1.8, Encino, California, 1991.

35. MITRE Corp., Report 11080, *Superworkstations for Terrain Visualization: A Prospectus*, by D. A. Southard, January 1991.

36. MITRE Corp., Report 90-78, Digital Technologies for Terrain Representation, by J. K. Rayson, January 1991.

37. Naval Ocean Research and Development Activity, NORDA R-194, *Design and Implementation of the Digital Vector Shoreline Data Format*, by M. C. Lohrenz, May 1988.

38. Naval Postgraduate School, Report 52-88-027, *Preliminary Work on the Command and Control Workstation of the Future*, by Harris, F. E., Yurchak, J. M., and Zyda, M. J., August 1988.

39. Naval Postgraduate School, Report 52-90-024, *An introduction to Object-Oriented Programming*, by Nelson, M. L., April 1990.

40. Nyberg, K. A., *Ada: Sources & Resources*, Grebyn Corporation, 1991.

41. Nyberg, K. A., Excerpt from the RWDB2 Users' Guide, 1.5 ed., Grebyn Corporation, 1992.

42. Pressman, R. S., *Software Engineering: A Practitioner's Approach*, 3d ed., McGraw-Hill, 1992.

43. Quercia, V., and O'Reilly, T., *X Window System User's Guide: OSF/Motif Edition*, v. 3, O'Reilly & Associates, Inc., January 1991.

44. Quick, D. A., *Developing Map-Based Graphics for the Theater War Exercise*, Master's Thesis, Air Force Institute of Technology, Air University, 1988.

45. Robinson, C. A., "Scattered Mobile Electronics Demand Trusted Reliability," *Signal*, v. 46, pp.31-32, January 1992.

46. Robinson, C. A., "Digital Technology Spawns Vivid Imaging Revolution," *Signal*, v. 46, pp. 21-24, May 1992.

47. Rome Laboratory, Air Force Systems Command, *Digital Cartographic Applications (DCA): Final Report*, Grumman Data Systems, 18 November 1991.

48. Science Applications International Corporation, *STARSAda/Xlib Bindings: X Window System, V11.R4*, Version 1, 7 November 1990

49. Seagle, J. P. and Belardo, S. "The Feature Chart: A Tool for Communicating the Analysis for a Decision Support System," *Information & Management*, v. 10 pp. 11-19, January 1986.

50. Shlaer, S., and Mellor, S. J., *Object-Oriented Systems Analysis: Modeling the World in Data*, Yourdon Press, 1988.

51. Software Productivity Solutions, Inc., *Classic-Ada User's Manual*, Indialantic, Florida, 1989.

52. Software Technology Support Center (STSC), *CrossTalk*, SofTech, Inc. March 1992.

53. Space and Warning System Center (SWSC), Directorate of Systems Development (SMD), Granite Sentry Development System, Peterson Air Force Base, Colorado, October 1991.

54. Strassmann, P. A., *The DOD Context for Integrated Computer Software Engineering*, I-CASE Bidders Conference Briefing, Director of Defense Information, Office of Assistant Secretary of Defense (Command, Control, Communications, and Intelligence), Montgomery Alabama, 31 March 1992.

55. Strassmann, P. A., *Implications of the Command and Control Functional Analysis and Consolidation Review Panel Report of 30 October, 1991*, Letter, Director of Defense Information, Office of the Assistant Secretary of Defense (Command, Control, Communications, and Intelligence), Department of Defense, 28 December 1991.

56. Stockwell, M. G., *A Graphical User Interface for The Low Cost Combat Direction System (LCCDS)*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1991.

57. Sun Microsystems, *SunOS Reference Manuals*, Version 4.1, 1990.

58. Tiburon Systems Inc., Report TIB 03961, *Advanced Tactical Workstation Computer Systems Operational Manual (ATW CSOM)*, Preliminary, Version 1.70, San Jose, California, No Date.

59. Tiburon Systems Inc., Report TIB 04820, *System Operator's Manual (SOM) for the B-52G Over-The-Horizon Airborne Sensors Information System (OASIS)*, San Jose, California, 28 September 1991.

60. Interview between Ms. Liz Sullinger, Tiburon Systems Inc., San Jose, California, and the author, 28 October 1991.

61. Transportable Application Environment (TAE) Plus, *User Interface Developer's Guide*, Version 5.1, National Aeronautics and Space Administration, Goddard Space Flight Center, April 1991

62. Tucker, W. H., *The X Map Projection Application (XMP)*, unpublished software, Tuckerware, Roswell, Georgia, 1992.

63. USAF Electronic Systems Division, *Positional Handbook, System Design and Analysis Support (SDAS) for the Granite Sentry System Program Office*, Draft, 1 May 1990.

64. USAF Rome Air Development Center, *Cartographic Applications for Tactical and Strategic Systems (CATSS) Functional Software Design*, RADC-TR-87-118, January 1988.

65. USAF Rome Air Development Center, *System / Segment Design Document for the Multi-Source Integrated Viewing System (MIVS)*, 9 January 1991.

66. United States Central Command, Command and Control Information System (C^2IS) Users' Manual, 30 September 1988.

67. United States General Accounting Office, *Programming Language: Status, Costs, and Issues Associated with Defense's Implementation of Ada*, GAO/IMTEC 89-9, March 1989.

68. Verdix Corporation, *Verdix Ada Development System (VADS)*, Version 6.0, Chantilly, Virginia, 1990

69. Yourdon, E., *Decline and Fall of the American Programmer*, Prentice Hall, 1992.

BIBLIOGRAPHY

Barkakati, N., *X Window System Programming*, SAMS, 1991.

Barnes, J. G. P., *Programming in Ada*, 3d ed., Addison-Wesley, 1989.

Barnes, P. D., *A Decision-Based Methodology for Object-Oriented Design*, Master's Thesis, Air Force Institute of Technology, Air University, December 1988.

Coskun, V., and Kesoglu, C., *A Software Prototype for a Command, Control, Communications, and Intelligence (C^3I) Workstation*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1990.

Foley, J. D., and others, *Computer Graphics: Principles and Practice*, Addison-Wesley, 1990.

Gonzalez, D. W., *Ada Programmer's Handbook and Language Reference Manual*, Benjamin/ Cummings, 1991.

Heckel, P., *The Elements of Friendly Software Design*, 2d ed., Sybex, 1991.

Skansholm, J., *Ada from the Beginning*, Addison-Wesley, 1988.

Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, 1987.

Sun, C. H., *Developing Portable User Interfaces for Ada Command Control Software*,M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1990.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center	2
Cameron Station	
Alexandria, VA 22304-6145	
Air Force Institute of Technology	1
CIRK	
Wright-Patterson Air Force Base, OH, 45433-6583	
Library, Code 0142	2
Naval Postgraduate School	
Monterey, CA 93943-5002	
C3 Academic Group, Code CC	1
Naval Postgraduate School	
Monterey, CA 93943-5000	
Captain B.K. Larson, USAF	2
1534 Southeast 59th Avenue	
Portland, OR 97215	
Captain P. D. Barnes, USAF	1
2062 CS	
PSC 20, BOX 3063	
APOAE 09260	
Dr. W. G. Kemple, Code OR/Ke	1
Naval Postgraduate School	
Monterey, CA 93943-5000	
Dr. Luqi, Code CS/Lq	1
Naval Postgraduate School	
Monterey, CA 93943-5000	
Major M.L. Nelson, USAF, Code CS/Ne	1
Naval Postgraduate School	
Monterey, CA 93943-5000	